APIs for Social Scientists: A Review (1.0)

22 November, 2021

Authors:
Paul C. Bauer, Jan Behnert, Lion Behrens, Lukas Isermann, Philipp Kadel, Melike N. Kaplan, Jana Klein, Barbara K. Kreis, Camille Landesvatter, Madleen Meier-Barthold, Pirmin Stöckle, Dean Lajic, Tymoteusz Ogłaza

Edited by:
Paul C. Bauer/Camille Landesvatter

We are happy about feedback or contributions to the repository: https://github.com/paulcbauer/apis_for_social_scientists_a_review.

If you would like to contribute further chapters please contact us under mail@paulcbauer.eu.

Contents

Tables

Figures

1 Introduction

The project *APIs for Social Scientists: A Review* is an outcome of the seminar *Computational Social Science* (CSS) taught at the University of Mannheim. While teaching the seminar we had trouble finding short reviews of APIs with quick R-code examples. Fortunately, almost everyone participating in the seminar was motivated enough to write a quick API review. The current document is available in PDF on OSF and as HTML. Hopefully, our ressource will be helpful future students that start to dive into CSS.

Below we review different data- and service-APIs that may be useful to social scientists. The chapters always include a simple R code example as well as references to social science research that has relied on them. The idea is to provide short reviews of max. 10 pages for the corresponding API with code to get you started. Each chapter follows a systematic set of questions:

- What data/service is provided by the API? (+ who provides it?)
- What are the prerequisites to access the API (e.g., authentication)?
- What does a simple API call look like?
- How can we access the API from R (httr + other packages)? * Are there social science research examples using the API?

1.1 Authentication

A lot of the APIs require that you authenticate with the API provider. The underlying script of this Review is written in such a way that it contains R chunks for authentication, however they will not be visible in the examples below (we only show placeholders for you to recognize at which step you will need to authenticate). These chunks in most cases make use of so-called keys in JSON format (e.g., service account key for Google APIs). However cloning the corresponding repository of this review will not result in giving you the keys, hence in order to replicate our API calls, you will have to generate and use your own individual key.

1.2 Packages

The code examples rely on different packages. It's probably easiest if you install all of them in one go. The `p_load()` function (`pacman` package) checks whether packages are installed. If not they are installed and loaded.

```
library(pacman)
pacman::p_load(
  dplyr
)
devtools::install_github("quanteda/quanteda.corpora")
```

1.3 Versioning

The objective is to have a document that is up to date. Hence, the versioning because it allows to produce new versions of the document in which outdated chapters are updated or excluded (e.g., if an API ceases to exist).

2 Google Natural Language API

Paul C. Bauer, Camille Landesvatter, Malte Söhren

2.1 Provided services/data

- *What data/service is provided by the API?*

The API is provided by Google. Google Cloud offers two Natural Language Products: AutoML Natural Language and Natural Language API. See here to read about which of the two products is the one more useful to you. Option 1, the Auto Machine Learning (ML) Natural Language allows you to train a new, custom model to either classify text, extract entities or detect sentiment. For instance, you could provide an already pre-labeled subset of your data which the API will then use to train a custom classifier. With this classifier at hand you could then classify and analyze further similar data of yours. This API review focuses on option 2, the Natural Language API. This API uses pre-trained models to analyze your data. Put differently, instead of providing only a pre-labeled subset of your data, here you normally provide the API your complete data which it will then analyze. The following requests are available:

- Analyzing Sentiment (`analyzeSentiment`)
- Analyzing Entities (`analyzeEntities`)
- Analyzing Syntax (`analyzeSyntax`)
- Analyzing Entity Sentiment (`analyzeEntitySentiment`)
- Classifying Content (`classifyText`)

A demo of the API that allows you to input text and explore the different classification capabilities can be found [here].

2.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

The prerequisite to access Google Natural Language API is a Google Cloud Project. To create this you need a Google account to log into the Google Cloud Platform. Before getting started, don't forget to enable the Natural Language API for your respective Google Cloud Project here (htt

. Additionally, if you are planning to request the Natural Language API from outside a Google Cloud environment (e.g., R and the googleLanguageR (https://cran.r-project.org/web/packages/googleLanguageR/vignettes/setup.html) package) you will be required to use a private (service account) key. This can be achieved by creating a service account (https://cloud.google.com/docs/authentication/production#create_service_account) which in turn will allow you to download your private key as a JSON file. Below we provide an example of how to authenticate from within the Google Cloud Platform (Cloud Shell + API key) and how to authenticate from within R (authentication via JSON key file). To create your API key for authentication from within the GCP, go to APIs & Services > Credentials (https://console.cloud.google.com/apis/credentials).

2.3 Simple API call

- *What does a simple API call look like?*

Below an expample of a simple API call via Google Cloud Shell

- To activate your Cloud Shell, inspect the upper right-hand corner of your Google Cloud Platform Console and click the icon called “Activate Shell”. Google Cloud Shell is a command line environment running in the cloud.
- Via the Cloud Shell command line, add your individual API key to the environment variables, so it is not required to be called for each request.

```
export API_KEY=<YOUR_API_KEY>
```

- Via the built-in Editor in Cloud Shell create a JSON file (call it for instance ‘request.json’) with the text that you would like to perform analysis on. Consider that text can be uploaded in the request (shown here) or integrated with Cloud Storage. Supported types of your text are PLAIN_TEXT (shown here) or HTML.

```
{
  "document":{
    "type":"PLAIN_TEXT",
    "content":"Enjoy your vacation!"
  },
```

```
  "encodingType": "UTF8"       }
```

- For sending your data, pass a curl command to your Cloud Shell command line where you refer (via @) to your request.json file from the previous step.

```
curl     "https://language.googleapis.com/v1/documents:analyzeEntities?
         key=${API_KEY}"     -s     -X     POST     -H     "Content-Type:
         application/json" --data-binary @request.json
```

- Depending on to which endpoint you send the request (here: analyzeEntities) you will receive your response with many insights into your text data.

2.4 API access

- *How can we access the API from R (httr + other packages)?*

Since the input we provide to the API can go beyond a single word/search term we will directly use the corresponding R package.

Example using R-Package 'googleLanguageR':

In this small example we demonstrate how to.. * .. authenticate with your Google Cloud Account within R * .. how to analyze the syntax of exemplary twitter data (we are using twitter data from two popular german politicians, which we (via the Google Translation API) beforehand also translated to english) * .. how to extract terms that are nouns only .. plot your nouns in a word cloud

```
# Load packages
library(tidyverse)
library(googleLanguageR)
library(tidytext)
library(ggwordcloud)
library(quanteda.corpora)

# Authentication (through your service account's JSON key
        file)
gl_auth("./keys/paul-css-seminar-2021-a1e75382ae2c.json")

# Load data
```

```
guardian_corpus                                                    <-
         quanteda.corpora::download("data_corpus_guardian")

texts <- guardian_corpus[["documents"]][["texts"]]
texts <- texts[1:20]
df <- as.data.frame(texts)
df <- tibble::rowid_to_column(df, "ID")
```

```
# Call the API via the function 'gl_nlp()' and specify your
         quantity of interest (here: analyzeSyntax)
data_syntax <- gl_nlp(df$texts, nlp_type = "analyzeSyntax")
```

Depending on what specific argument you used (here: analyzeSyntax) a list with information on different characteristics of your text is returned, e.g., sentences, tokens, tags (noun, verb, etc.) And the output of "analyzeSyntax" most importantly stores two results: "content" (contains the token) and "tag" (contains the tag, e.g., verb or noun)

```
# Have a look
#head(data_syntax[["tokens"]][[1]][,1:5])

# Add dataframe with tokens syntax to original dataframe
df$syntax_tokens <- data_syntax[["tokens"]]

# Now we are only interested in tokens that are marked as
         "nouns"
# Filter out nouns only
df <- df %>%
  mutate(tokens_nouns = map(syntax_tokens,
                            ~ filter(., tag == "NOUN")))

# Create the data for the plot
data_plot <- df %>%
  # only keep content variable
  mutate(tokens_nouns = map(tokens_nouns,
                            ~ select(., content))) %>%
  # Write tokens in all rows into a single string
  unnest(tokens_nouns) %>% # unnest tokens
  # Unnest tokens
    unnest_tokens(output = word, input = content) %>% #
         generate a wordcloud
  anti_join(stop_words) %>%
```

```
  dplyr::count(word) %>%
  filter(n > 10) #only plot words that appear more than 10
         times
         # Visualize in a word clouddata_plot %>%
  ggplot(aes(label = word,                      size = n)) +
  geom_text_wordcloud() +
  scale_size_area(max_size = 10) +
  theme_minimal()
```

Figure 2.1: Wordcloud of nouns found within guardian articles

2.5 Social science examples

- *Are there social science research examples using the API?*

Generally, we have the impression that the usage of Google's Natural Language API is relatively unknown in NLP and among social scientists working with text data. However, we want to emphasize the usefulness and importance the usage of Google's NLP API could have in many research projects.

3 Google Translation API

Paul C. Bauer, Camille Landesvatter

3.1 Provided services/data

- *What data/service is provided by the API?*

The API is provided by Google. Google's Translation API translates texts into more than one hundred languages (https://cloud.google.com/translate/docs/languages). Note that the approach via the API is a lot more refined than the free version (https://translate.google.com/?hl=de) on Google's translation website and of course comes in very useful if text in large scale needs to be translated (possibly with longer and more complex content or syntax). For instance, you can choose to specify a certain model to further improve the translation (Neural Machine Translation vs. Phrase-Based Machine Translation (https://cloud.google.com/translate/docs/basic/translating-text#translate_translate_text-python)).

The API limits in three ways: characters per day, characters per 100 seconds, and API requests per 100 seconds. All can be set in the API manager (https://console.developers.google.com/apis/api/translate.googleapis.com/quotas) of your Google Cloud Project.

Consider that besides the Translation API which we present here, Google provides two further APIs for translation: AutoML Translation and the Advanced Translation API (see here (https://cloud.google.com/translate/?utm_source=google&utm_medium=cpc&utm_campaign=emea-de-all-de-dr-bkws-all-all-trial-e-gcp-1010042&utm_content=text-ad-none-any-DEV_c-CRE_170514365277-ADGP_Hybrid%20%7C%20BKWS%20-%20EXA%20%7C%20Txt%20~%20AI%20%26%20ML%20~%20Cloud%20Translation%23v3-KWID_43700053282385063-kwd-74703397964-userloc_9042003&utm_term=KW_google%20translator%20api-NET_g-PLAC_&gclid=CjwKCAjw_JuGBhBkEiwA1xmbRZOxg7QzGmhTHWseHFN_V0AI_Xlf8wZVBfX9EURtitWDbe2dLcTWIxoCjj0QAvD_BwE&gclsrc=aw.ds#section-4) for a short comparison).

3.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

To access and to use the API the following steps are necessary:

- Create a google account (https://www.google.com/account/about/) (if you do not already have one)
- With this google account login to the google cloud platform (https://cloud.google.com/) and create a Google Cloud Project
- Within this Google Cloud Project enable the Google Translation API
- For authentication you will need to create an API key (which you additionally should restrict to the Translation API). If however, you are planning to request the Natural Language API from outside a Google Cloud environment (e.g., R) you will be required to use a private (service account) key. This can be achieved by creating a service account which in turn will allow you to download your private key as a JSON file (we show an example below).

3.3 Simple API call

- *What does a simple API call look like?*

Below an expample of a simple API call via Google Cloud Shell

- To activate your Cloud Shell, inspect the upper right-hand corner of your Google Cloud Platform Console and click the icon called “Activate Shell”. Google Cloud Shell (https://cloud.google.com/shell/#how_do_i_get_started) is a command line environment running in the cloud.
- Via the built-in Editor in Cloud Shell create a JSON file (call it for instance ‘request.json’) with the text that you would like to perform analysis on. Consider that text can be uploaded in the request (shown here) or integrated with Cloud Storage. Supported types of your text are PLAIN_TEXT (shown here) or HTML.

```
{
  "q": ["To administer medicine to animals is frequently a
         very difficult matter, and yet sometimes it’s
         necessary to do so"],
"target": "de"
}
```

- For sending your data, pass a curl command to your Cloud Shell command line where you refer (via @) to your request.json file from the previous step.
- Don't forget to insert your individual API key (alternatively, you could define it beforehand via a variable in your environment -> see example in the API call for Google's NLP API later in this document).

```
curl "https://translation.googleapis.com/language/translate/v2?key=APIKEY" -s -X POST -H "Content-Type: application/json" --data-binary @request.json
```

- Depending on to which endpoint you send the request (here: analyzeEntities) you will receive your response with many insights into your text data.

3.4 API access

- *How can we access the API from R (httr + other packages)?*

Since the input we provide to the API can go beyond a single word/search term we will directly use the corresponding R package.

Example using R-Package 'googleLanguageR':

In this small example we demonstrate how to.. *.. authenticate with your Google Cloud Account within R

*.. how to translate an exemplary sentence

*.. how to bind the results or the translation to your original dataframe

```
# Load packages
library(tidyverse)
library(googleLanguageR)

# Authentication (through your service account's JSON key file)
gl_auth("./keys/trustme-312210-41f50915e801.json")
```

We will translate the text (see translation API) and store it in `text_en` and use the English translations below:

```
# Create data frame with text
# here we only include one exemplary sentence; for your own
        project use a complete vector containing text data
        from within your dataset

df_original <- data.frame(text = "To administer medicine to
        animals is frequently a very difficult matter, and
        yet sometimes it's necessary to do so")

df_original$text <- as.character(df_original$text)
```

```
# Call the API via the function 'gl_translate() ' and
        specify your target language
df_translate <- gl_translate(df_original$text, target =
        "de")
```

The API provides three outputs (each as one column): translatedText (contains the translation), detectedSourceLangauge (contains a label for the original language being detected) and text (contains the original text).

```
# View some translations
head(df_translate)
```

```
## # A tibble: 1 x 3
##   translatedText                    detectedSourceLan~ text
##   <chr>                             <chr>              <chr>
## 1 Tieren Medikamente zu verabreich~ en                 To administer medicin
```

```
# Add translation to your original data
df_orginal <- bind_cols(df_original,
                                          df_translate %>%
        select(translatedText))
```

3.5 Social science examples

- *Are there social science research examples using the API?*

Generally, we are not aware of research that made explicit usage of Google's Translation API. However, we assume that the API (or at least Google's Free Translation Service) is involved in many research projects. One stream of research where (Google's) Translation Services will appear several times, is presented in a study by Prates, Avelar and Lamb (2020): Assessing gender bias in machine translation: a case study with Google Translate (https://arxiv.org/abs/1809.02208). The paper examines important topics of social science research with regards to social inequality. Further, we are convinced that making use of automated translation (as well as converting speech to text eventually in combination with translation; see next chapter) can be of great advantage for all kinds of qualitative or mixed-methods research projects. For instance, automated translation could be very useful for easily and very reliably translating data from qualitative interviews or other (field) experiments or observational data (where data additionally is stored in a foreign language). Also consider the other way around where data is available in the principal language of the research project but some of the textual data has to be communicated and presented in (for instance) english language.

4 CrowdTangle API

Lion Behrens and Pirmin Stöckle

CrowdTangle is a public insights tool, whose main intent was to monitor what content overperformed in terms of interactions (likes, shares, etc.) on Facebook and other social media platforms. In 2016, CrowdTangle was acquired by Facebook that now provides the service.

4.1 Provided services/data

- *What data/service is provided by the API?*

CrowdTangle allows users to systematically follow and analyze what is happening with public content on the social media platforms of Facebook, Twitter, Instagram and Reddit. The data that can be assessed through the CrowdTangle API consists of any post that was made by a public page, group or verified public person who has ever acquired more than 110,000 likes since the year 2014 or has ever been added to the list of tracked public accounts by any active API user. If a new public page or group is added, data is pulled back from day one.

Data that is tracked:

- Content (the content of a post, including text, included links, links to included images or videos)
- Interactions (count of likes, shares, comments, emoji-reactions)
- Page Followers
- Facebook Video Views
- Benchmark scores of all metrics from the middle 50% of posts in the same category (text, video) from the respective account

Data that is not tracked:

- Comments (while the number of comments is included, the content of the comments is not)
- Demographics data
- Page reach, traffic and clicks
- Private posts and profiles
- Ads only appear in the ad library (which is public), boosted content cannot differentiated from organic content

CrowdTangle's database is updated once every fifteen minutes and comes as time-series data which merges the content of a post on one of the included platforms (a text post, video, or image) alongside aggregate information on the post's views, likes and interactions.

When connecting to the user interface via the CrowdTangle website, the user can either manually set up a list of pages of interest whose data should be acquired. Alternatively, one can choose from an extensive number of pre-prepared lists covering a variety of topics, regions, or socially and politically relevant events such as inaugurations and elections. Data can be downloaded from the user interface as csv files or as json files via the API.

4.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

Full access to the CrowdTangle API is only given to Facebook partners who are in the business of publishing original content or fact-checkers as part of Facebook's Third-Party Fact-Checking program. From 2019, the CrowdTangle API and user interface is also available for academics and researchers in specific fields. Currently, this prioritization includes research on one of the following fields: misinformation, elections, COVID-19 racial justice, well-being. To get access to CrowdTangle, a formal request has to be filed via an online form, asking for a short description of the research project and intended use of the data.

As a further restriction, CrowdTangle currently only allows academic staff, faculty and registered PhD students permission to obtain a CrowdTangle account. This does not include individuals enrolled as students at a university unless they are employed as research assistants. Also, certain access policies differ between academics and the private sector. Usage of CrowdTangle for research purposes does currently not provide access to any content posted on Reddit given that data is retrieved via the Application Programming Interface. Content from Reddit is open to every registered user only when navigating through the company's dynamic user interface that does not imply usage of any scripting language. Finally, the CrowdTangle API requires researchers to log in using an existing Facebook account. Overall, access to the API is quite restrictive, both because of the prioritization of certain research areas, and because the access request will be decided individually so that an immediate access is not possible. If access is

granted, CrowdTangle provides quite extensive onboarding and training resources to use the API. Replicability

Access to CrowdTangle is gated and Facebook does not allow data from CrowdTangle to be published. So researchers can publish aggregate results from analyses on the data, but not the original data, which might be problematic for the replicability of research conducted with the API. A possible workaround is that you can pull ID numbers of posts in your dataset, which can then be used by anyone with a CrowdTangle API access to recreate your dataset. CrowdTangle also provides some publicly available features such as a Link Checker Chrome Extension, allowing users to see how often a specific link has been shared on social media, and a curated public hub of Live displays, giving insight about specific topics on Facebook, Instagram and Reddit.

4.3 Simple API call

- *What does a simple API call look like?*

All requests to the CrowdTangle API are made via GET to https://api.crowdtangle.com/ In order to access data, users log in on the website with their Facebook account and acquire a personalized token. The CrowdTangle API expects the API token to be included in each query. With one of these available endpoints, each of which comes with a set of specific parameters:

GET /posts	Retrieve a set of posts for the given parameters.
GET /post	Retrieves a specific post.
GET /posts/search	Retrieve a set of posts for the given parameters and search terms.
GET /leaderboard	Retrieves leaderboard data for a certain list or set of accounts.
GET /links	Retrieve a set of posts matching a certain link.
GET /lists	Retrieve the lists, saved searches and saved post lists of the dashboard associated with the token sent in.

- A simple example

Which party or parties posted the 10 most successful Facebook posts this year? On the user interface, I created a list of the pages of all parties currently in the German Bundestag. We want to find out which party or parties posted the 10 most successful posts (i.e. the posts with the most interactions) this year.

The respective API call looks like that: https://api.crowdtangle.com/posts?token=token&listIds=listIDs&sortBy=total_interactions&startDate=2021-01-01&count=10 (https://api.crowdtangle.com/posts?token=token&listIds=listIDs&sortBy=total_interactions&startDate=2021-01-01&count=10%5D)

Where token is the personal API key, and listIDs is the ID of the list created with the user interface. Here, we sortBy total interactions with the startDate at the beginning of this year and the output restricted to count 10 posts.

4.4 API access

- *How can we access the API from R (httr + other packages)?*

Instead of typing the API request into our browser, we can use the httr package's GET function to access the API from R.

```
# Option 1: Accessing the API with base "httr" commands
library(httr)

ct_posts_resp <- GET("https://api.crowdtangle.com/posts",
     query=list(token = token, # API key has to be included
          in every query
                listIds = listIds, # ID of the created list
          of pages or groups
                sortBy = "total_interactions",
                startDate = "2021-01-01",
                count = 10))

ct_posts_list <- content(ct_posts_resp)
class(ct_posts_list) # verify that the output is a list

# List content
str(ct_posts_list, max.level = 3) # show structure & limit
          levels
```

```
# with some list operations we can get a dataframe with the
        account name and post date of the 10 posts with the
        most interactions in 2021 among the pages in the
        list
list_part <- rlist::list.select(ct_posts_list$result$posts,
        account$name, date)
rlist::list.stack(list_part)
```

Alternatively, we can use a wrapper function for R, which is provided by the RCrowdTangle package available on github (https://github.com/cbpuschmann/RCrowdTangle). The package provides wrapper functions for the /posts, /posts/search, and /links endpoints. Conveniently, the wrapper function directly produces a dataframe as output, which is typically what we want to work with. As the example below shows, the wrapper function may not include the specific information we are looking for, however, as the example also shows, it is relatively straightforward to adapt the function on our own depending on the specific question at hand. To download the package from github, we need to load the devtools package, and to use the wrapper function, we need dplyr and jsonlite.

```
# Option 2: There is a wrapper function for R, which can be
        downloaded from github

library(devtools) # to download from github

install_github("cbpuschmann/RCrowdTangle")
library(RCrowdTangle)

# The R wrapper relies on jsonlite and dplyr
library(dplyr)
library(jsonlite)

ct_posts_df <- ct_get_posts(listIds, startDate = "2021-01-
        01", token = token)

#conveniently, the wrapper function directly produces a
        dataframe
class(ct_posts_df)

# to sort by total interactions we have to compute that
        figure because it is not part of the dataframe
ct_posts_df %>%
```

```
                        mutate(total_interactions           =
          statistics.actual.likeCount+statistics.actual.shareCount+
                                statistics.actual.commentCount+
          statistics.actual.loveCount+
          statistics.actual.wowCount+
          statistics.actual.hahaCount+
          statistics.actual.sadCount+
                                 statistics.actual.angryCount+
          statistics.actual.thankfulCount+
          statistics.actual.careCount) %>%
  arrange(desc(total_interactions)) %>%
  select(account.name, date) %>%         head(n=10)

# alternatively, we can adapt the wrapper function by
        ourselves to include the option to sort by total
        interactions
ct_get_posts <- function(x = "", searchTerm = "", language =
        "", types= "", minInteractions = 0, sortBy = "",
        count = 100, startDate = "", endDate = "", token =
        "")
{
  endpoint.posts <- "https://api.crowdtangle.com/posts"
    query.string <- paste0(endpoint.posts, "?listIds=", x,
        "&searchTerm=", searchTerm, "&language=", language,
        "&types=",        types,        "&minInteractions=",
        minInteractions,  "&sortBy=",  sortBy,  "&count=",
        count,   "&startDate=",   startDate,   "&endDate=",
        endDate, "&token=", token)
    response.json <- try(fromJSON(query.string), silent =
        TRUE)
  status <- response.json$status
  nextpage <- response.json$result$pagination$nextPage
     posts  <-  response.json$result$posts  %>%  select(-
        expandedLinks, -media) %>% flatten()
  return(posts)       }
ct_posts_df     <-     ct_get_posts(listIds,     sortBy     =
        "total_interactions",  startDate  =  "2021-01-01",
        token = token)
        ct_posts_df %>%
  select(account.name, date) %>%         head(n=10)
```

4.5 Social science examples

- *Are there social science research examples using the API?*

A common use case is to track the spread of specific links containing misinformation, e.g. conspiracy around the connection of COVID-19 and 5G (Bruns et al 2020). Berriche and Altay (2020) provide an in-depth analysis of a specific page involved in online health misinformation and investigate factors driving interactions with the respective posts. They find that users mainly interact to foster social relations, not to spread misinformation. CrowdTangle has also been used to study changes in the framing of vaccine refusal by analyzing content of posts by pages opposing vaccinations over time (Broniatowksi et al 2020). Another approach is to monitor political communication of political actors, specifically in the run-up to elections. Larsson (2020) investigates a one-month period before the 2018 Swedish election and finds that right-wing political actors are more successful than mainstream actors in engaging their Facebook followers, often using sensational rhetoric and hate-mongering.

4.5.1 References

Berriche, M., & Altay, S. (2020). Internet users engage more with phatic posts than with health misinformation on Facebook. Palgrave Communications, 6(1), 1-9.

Broniatowski, D. A., Jamison, A. M., Johnson, N. F., Velasquez, N., Leahy, R., Restrepo, N. J., … & Quinn, S. C. (2020). Facebook Pages, the “Disneyland” Measles Outbreak, and Promotion of Vaccine Refusal as a Civil Right, 2009–2019. American Journal of Public Health, 110(S3), S312-S318.

Bruns, A., Harrington, S., & Hurcombe, E. (2020). ‘covid19? ‘Corona? 5G? or both?’: the dynamics of COVID-19/5G conspiracy theories on Facebook. Media International Australia, 177(1), 12-29.

Larsson, A. O. (2020). Right-wingers on the rise online: Insights from the 2018 Swedish elections. New Media & Society, 22(12), 2108-2127.

5 Google Places API

Lukas Isermann and Clara Husson

5.1 Provided services/data

- *What data/service is provided by the API?*

The following five requests are available: Place Search, Place Details, Place Photos, Place Autocomplete and Query Autocomplete. Place Search returns a list of places along with summary information about each place based on a user's location (by proximity) or search string. Once you find a place_id from a Place Search, you can request more details about a particular place by doing a Place Details request. A Place Details request returns more detailed information about the indicated place such as its complete address, phone number, user rating and reviews. Place Photos provides access to the millions of place-related photos stored in Google's Place database. When you get place information using a Place Details request, photo references will be returned for relevant photographic content. Find Place, Nearby Search, and Text Search requests also return a single photo reference per place, when relevant. Place Autocomplete automatically fills in the name and/or address of a place as users type. Query Autocomplete service provides a query prediction for text-based geographic searches, by returning suggested queries as you type.

Note: You can display Places API results on a Google Map, or without a map but it is prohibited to use Places API data on a map that is not a Google map.

5.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

The prerequisites to access Google Places API are a Google Cloud project (to create it you need a Google account to log into the Google Cloud Platform) and an API Key. Before creating your API Key, don't forget to enable Places API! To create your API key, go the APIs & Services > Credentials > API key page.

5.3 Simple API call

- *What does a simple API call look like?*

The API provides different searches and services that can be accessed via HTTP Urls. These Urls requests all take the same general form and pattern:

https://maps.googleapis.com/maps/api/place/service/output?parameters

Here, service can take the inputs findplacefromtext for find place requests, nearbysearch to look for nearby places, details for a request of place details and more. output may take the value json or xml, dependent on the requested output format.

Furthermore, certain parameters are required for all requests. Most importantly, every request must entail a key parameter, indicating the API key. Second, all search places requests take an input parameter that identifies the search target and an inputtype parameter that identifies the type of input given in the input parameter. For place requests, the inputtype parameter can take the values textquery and phonenumber.

Nearby requests take a location parameter setting longitude and latitude of the requested place as well as a radius parameter. Detail request, however, take a mandatory parameter place_id, which indicates the place for which the details are requested.

Additionally, different optional parameters can be used. These entail a language parameter, a fields parameter indicating the types of place data to return and more.

An examples for an API request for pizza places in Mannheim can look like this:

https://maps.googleapis.com/maps/api/place/textsearch/xml?query=pizza&location=49.487459,8.466039&radius=5000&key=YOUR_API_KEY

5.4 API access

- *How can we access the API from R (httr + other packages)?*

Instead of typing the API request into our browser, we can use the httr package's GET function to access the API from R.

```
# Option 1: Accessing the API with base "httr" commands
library(httr)

library(httr)
key <- "YOURAPIKEY"

res<-
          GET("https://maps.googleapis.com/maps/api/place/textsearch/json?",
           query = list(
            query = "pizza",
            location = "49.487459,8.466039",
            radius = 5000,
            key = key
          ))
```

Alternatively, we can use a wrapper function for R provided by the R-Package googleway.

```
# Option 2: Accessing the API with googleway

library(ggplot2)
library(tidyverse)
library(googleway)
key <- "YOURAPIKEY"
set_key(key)

#  Request  'Mannheim'  to  get  latitude  and  longitude
          information
location <- google_places("Mannheim")

# Save latitude and longitude information in vector
location <- location$results$geometry
location <- c(location$location$lat, location$location$lng)

# Google places request with googleway
pizza <- google_places("Pizza", location = location, radius
          = 5000, place_type = "food")

# Plot rankings as barplot
```

```
pizza$results %>%          ggplot() +
  geom_col(aes(x = reorder(name, rating), y = rating)) +
  geom_text(aes(x = reorder(name, rating), y = rating,
                        label = paste0(user_ratings_total, "
        ratings",
               angle = 90, hjust = 0), size = 3) +
  ylab("Average Rating")+         xlab("") +
  ggtitle("Pizza Places in Mannheim by Rating") +
  theme_minimal() +         theme(
       axis.text.x = element_text(angle = 90, size = 8,
         hjust=0.95,vjust=0.2)
  )               # Plot places to google map
map <- google_map(location = location)
add_markers(map, data = pizza$results$geometry$location)
```

5.5 Social science examples

- *Are there social science research examples using the API?*

In his study “Using Google places data to analyze changes in mobility during the COVID-19 pandemic”, Markus Konrad looked at the “popular times” data provided by Google Places to measure the effect of social distancing effort on mobility.

Markus Konrad on the Wissenschaftszentrum Berlin für Sozialforschung (WZB) Data Science Blog in 2020: “Using Google places data to analyze changes in mobility during the COVID-19 pandemic”

https://datascience.blog.wzb.eu/2020/05/11/using-google-places-data-to-analyze-changes-in-mobility-during-the-covid-19-pandemic/ (https://datascience.blog.wzb.eu/2020/05/11/using-google-places-data-to-analyze-changes-in-mobility-during-the-covid-19-pandemic/)

6 Google Speech-to-Text API

Camille Landesvatter

6.1 Provided services/data

- *What data/service is provided by the API?*

Google's Speech-to-Text API allows you to convert audio files to text by applying powerful neural network models. Audio content can be transcribed in real time and of course (and possibly of higher relevance for social science research) from stored files.

The API currently recognizes more than 125 languages (https://cloud.google.com/speech-to-text/docs/languages). It supports multiple audio formats, and audio files can either be transcribed directly (if the content does not exceed 60 seconds) or perform asynchronous requests for audio files longer than 60 seconds.

A demo of the API that allows you to record text via your microphone (or to upload an audio file) and explore the transcript can be found here (https://cloud.google.com/speech-to-text#section-2). . Also consider that there is a Text-to-Speech API (https://cloud.google.com/text-to-speech) offered by Google.

6.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

To access and to use the API the following steps are necessary:

- Create a google account (https://www.google.com/account/about/) (if you do not already have one)
- With this google account login to the google cloud platform (https://cloud.google.com/) and create a Google Cloud Project
- Within this Google Cloud Project enable the Google Speech-to-text API
- For authentication you will need to create an API key (which you additionally should restrict to the Translation API). If however,

you are planning to request the Natural Language API from outside a Google Cloud environment (e.g., R) you will be required to use a private (service account) key. This can be achieved by creating a service account which in turn will allow you to download your private key as a JSON file (we show an example below).

6.3 Simple API call

- *What does a simple API call look like?*

Note. For both Google's Translation API as well as Google's Natural-Language API, in this review we demonstrate an example for a simple API call via the Google Cloud Shell. In principle (and in a very similar procedure) this can be achieved for the Speech-to-Text API. However, your audio file will need some pre-processing. Audio data (such as the exemplary wav-file we are using here) is binary data. To make your REST request (via the Google Cloud Shell) however JSON is used. JSON eventually does not support binary data, which is why you will have to transform your binary audio file into text using Base64 (https://en.wikipedia.org/wiki/Base64) encoding. Also read this documentation (https://cloud.google.com/speech-to-text/docs/base64-encoding#linux) from the Google Website. If you enter audio data which is not Base64 encoded, the Google Cloud Shell will give you an error 400 stating that Base64 decoding failed for your (wav-) file. Nevertheless, in the box below we will provide the basic structure of the request.

Below an example for a simple API call via Google Cloud Shell

- To activate your Cloud Shell, inspect the upper right-hand corner of your Google Cloud Platform Console and click the icon called "Activate Shell". Google Cloud Shell (https://cloud.google.com/shell/#how_do_i_get_started) is a command line environment running in the cloud.

- Via the built-in Editor in Cloud Shell create a JSON file (call it for instance 'request.json'). You can either upload your audio file directly via the Google Cloud Shell (search for the three-dotted "More" menu in the Shell and select "Upload file"), alternatively audio content can be integrated with Cloud Storage.

- The wav.file we uploaded for this example is an exemplary wav.file that comes along with "GoogleLanguageR"-package in

R (see below for an instruction)

```
{
  "audio": {
    "content": "woman1_wb"
  },
  "config": {
    "enableAutomaticPunctuation": true,
    "encoding": "LINEAR16",
    "languageCode": "en-US",
    "model": "default"
  }
}
```

- For sending your data, pass a curl command to your Cloud Shell command line where you refer (via @) to your request.json file from the previous step.
- Don't forget to insert your individual API key (alternatively, you could define it beforehand via a variable in your environment -> see example in the API call for Google's NLP API later in this document).

```
curl     "https://speech.googleapis.com/v1p1beta1/speech:recognize?
         key=APIKEY"     -s     -X     POST     -H     "Content-Type:
         application/json" --data-binary @request.json
```

6.4 API access

- *How can we access the API from R (httr + other packages)?*

Example using R-Package 'googleLanguageR':

In this small example we demonstrate how to..

*.. authenticate with your Google Cloud Account within R

*.. how to import an exemplary audiofile from the "GoogleLanguageR" package

*.. how to transcribe this audio file and calculate a confidence score

For the usage of further arguments, also read the `gl_speech()` documentation (https://cran.r-project.org/web/packages/googleLanguageR/googleLanguageR.pdf) and this (https://cran.r-project.org/web/packages/googleLanguageR/vignettes/speech.html) vignette.

```
# Load packages
library(tidyverse)
library(googleLanguageR)

# Authentication (through your service account's JSON key
        file)
gl_auth("./keys/trustme-312210-41f50915e801.json")
```

We will now get a sample source file which comes along with the `googleLanuageR` package. The transcript of this file is: “To administer medicine to animals is frequently a very difficult matter, and yet sometimes it’s necessary to do so”.

```
test_audio   <-   system.file("woman1_wb.wav",   package   =
        "googleLanguageR")
```

```
# Call the API via the function 'gl_speech()' and specify
        your quantities of interest
# here we specify the audio_source (this can either be a
        local file or a Google Cloud Storage URI) and the
        languageCode (language spoken in your audio file)
audio_data <- gl_speech(test_audio, languageCode = "en-GB")
```

The API returns a list of two dataframes: * transcript and timings

```
dimnames(audio_data$transcript)
```

```
## [[1]]
## [1] "1"
##
## [[2]]
## [1] "transcript"   "confidence"   "languageCode" "channelTag"
```

The timings dataframe stores timestamps on when specific words were recognised. The transcript dataframe among other information provides the transcript as well as a confidence score. We can see that the transcript misses one term (“a”) and indicates its confidence with a score close to 1.0

```
audio_data$transcript$transcript
```

```
## [1] "to administer medicine to animals is frequently very difficult matter a
```

```
audio_data$transcript$confidence #0.92
```

```
## [1] "0.9151854"
```

6.5 Social science examples

- *Are there social science research examples using the API?*

Similar to our note on social science research examples for Google’s Translation API, we are not aware of research that made explicit usage of Google’s Speech-to-text API. However, and especially in combination with the Translation API, we are convinced that speech-to-text conversion can be of great advantage for all kinds of qualitative or mixed-methods research projects.

7 Instagram Graph API

Philipp Kadel

7.1 Provided services/data

- *What data/service is provided by the API?*

The Instagram Graph API is provided by Facebook. There are two main APIs for Instagram, the Instagram Basic Display API and the Instagram Graph API. The latter is described in the following.

The API can be used to get and manage published photos, videos, and stories as well as getting basic data about other Instagram Business users and Creators. It is also possible to moderate comments and their replies and to measure media and profile interaction. Photos and videos can be published directly from the API. It can also be used to discover hashtagged media and @mentions.

For photos and videos different metrics can be obtained:

- engagement – Total number of likes and comments on the media object.
- Impressions – Total number of times the media object has been seen.
- Reach – Total number of unique accounts that have seen the media object.
- Saved – Total number of unique accounts that have saved the media object.
- Video_views – (Videos only) Total number of times the video has been seen. Returns 0 for videos in carousel albums.

Likewise, there are several metrics about stories that are provided by the API:

- Exits – Number of times someone exited the story.
- Impressions – Total number of times the story has been seen.
- Reach – Total number of unique accounts that have seen the story.
- Replies – Total number of replies to the story.

- Taps_forward – Total number of taps to see this story's next photo or video.

Sources:

https://developers.facebook.com/docs/instagram-api?locale=en_US

https://jamiemaguire.net/index.php/2019/06/22/instagram-graph-api-part-2-fetching-data-with-the-insights-api/

7.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

For most endpoints you need an Instagram Business Account, a Facebook Page that is connected to that account, a Facebook Developer Account and a Facebook App with Basic settings configured. Facebook provides a tutorial for setting this up here (https://developers.facebook.com/docs/instagram-api/getting-started).

7.3 Simple API call

- *What does a simple API call look like?*

Below you can find expamples of simple API calls for the Instagram Graph API

Get Fields and Edges on an IG Media. Fields can be e.g., "caption", "comments_count", "like_count", or "timestamp".

- GET ("https://graph.facebook.com/v10.0/{ig-media-id} ?fields= {fields} &access_token={access-token}")

Example:

```
GET                                                    ("https://graph.facebo
          fields=id,media_type,media_url,owner,timestamp&access_token=IGQVJ..."
```

Response:

```
{
  "id": "17895695668004550",
```

```
   "media_type": "IMAGE",
           "media_url":     "https://fb-s-b-a.akamaihd.net/h-ak-
          fbx/t51.2885-
          9/21227247_1640962412602631_3222510491855224832_n.jpg?
          _nc_log=1",
   "owner": {           "id": "17841405822304914"
   },         "timestamp": "2017-08-31T18:10:00+0000"
```

Return Fields and Edges on an IG Hashtag. Field can be the name of the hashtag without the “#” symbol or a hashtag ID.

- GET (“https://graph.instagram.com/{ig-hashtag-id} ?fields= {fields} &access_token={access-token}”)

Example:

```
GET (“https://graph.facebook.com/17841593698074073
?fields=id,name
&access_token=EAADd...”)
```

Response:

```
{ "id": "17841593698074073",
  "name": "coke" }
```

Get fields and edges on an Instagram Business or Creator Account. Fields can be e.g., “biography”, “id”, “followers_count”, or “media_count”.

- GET (“https://graph.facebook.com/v10.0/{ig-user-id}?fields= {fields} &access_token={access-token}”)

Example:

```
GET                                                    (“https://graph.faceboo
         fields=biography%2Cid%2Cusername%2Cwebsite&access_token=EAACwX...”)
```

Response:

```
{  "biography": "Dino data crunching app",
  "id": "17841405822304914",
```

```
   "username": "metricsaurus",
   "website": "http://www.metricsaurus.com/" }
```

Source: https://developers.facebook.com/docs/instagram-api/reference

7.4 API access

- *How can we access the API from R (httr + other packages)?*

The httr package can be used to access the Instagram Graph API. There used to be a instaR package but it was made for the old Instagram API and can not be used anymore. The FBinsightsR package provides access to the Insights API. Its fbins_insta function can be used to collect Instagram insights. Detailed information can be found here:

https://github.com/Deducive/FBinsightsR/blob/master/R/functions.R

https://stackoverflow.com/questions/54488186/instagram-api-in-r-deprecated-need-to-use-graph-facebook-com

7.5 Social science examples

- *Are there social science research examples using the API?*

In their study, Ferwerda, Schedl, and Tkalcic (2015) tried to infer personality traits from the way users take pictures and apply filters to them. The authors found distinct picture features (e.g., hue, brightness, saturation) that are related to personality traits. Brown, Bendig, Fischer, Goldwich, Baumeister, and Plener (2019) investigated the link between acute suicidality and language use as well as activity on Instagram. Differences in activity and language use on Instagram were not associated with acute suicidality. The goal of a study by Hosseinmardi, Mattson, Rafiq, Han, Lv, and Mishr (2015) was to automatically detect and predict incidents of cyberbullying on Instagram. Based on a sample data set consisting of Instagram images and their associated comments, media sessions were labeled for cyberbullying. Associations are investigated between cyberbullying and a host of features such as cyber aggression, profanity, social graph features, temporal commenting behavior, linguistic content, and image content.

Brown, R. C., Bendig, E., Fischer, T., Goldwich, A. D., Baumeister, H., & Plener, P. L. (2019). Can acute suicidality be predicted by Instagram data? Results from qualitative and quantitative language analyses. PloS

one, 14(9), e0220623. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0220623

Ferwerda, B., Schedl, M., & Tkalcic, M. (2015, September). Predicting personality traits with instagram pictures. In Proceedings of the 3rd Workshop on Emotions and Personality in Personalized Systems 2015 (pp. 7-10). https://dl.acm.org/doi/abs/10.1145/2809643.2809644?casa_token=AmeFxOkVYT8AAAAA:9jFd40K-ZPgpAS2s6vo1Gn_02SIlQeEvsRlgF750zBKz_5LSqqgornnDc2eAObmy2H5y4a8nm1-bwQ

Hosseinmardi, H., Mattson, S. A., Rafiq, R. I., Han, R., Lv, Q., & Mishr, S. (2015). Prediction of cyberbullying incidents on the Instagram social network. arXiv preprint arXiv:1508.06257. https://arxiv.org/abs/1508.06257