APIs for Social Scientists: A Review

Version: 17 June, 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

If this document is useful for you please cite the whole document or single chapters as follows: …

We also happy about any 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

Application programming interfaces (APIs) are an increasingly relevant tool for researchers to access data or services of various private or governmental platforms. Below we review different data- and service-APIs that may be useful to social scientists. We provide an overview over different APIs (see the APIs listed in the TOC below) following a systematic set of questions:

- What data/service is provided by the API?
- What are the prerequisites to access the API (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?

The project *APIs for Social Scientists: A Review* is an outcome of the seminar *Computational Social Science* (CSS) taught at the University of Mannheim during which we had trouble finding short reviews of APIs with quick R-code examples. Fortunately, everyone participating in the seminar was motivated enough to write a quick API review. Hopefully, our ressource will be helpful future students that start to dive into CSS. The chapters below 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.

- Data vs. machine learning APIS
- Problem of replicability for ML APIs

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 (https://console.cloud.google.com/marketplace/product/google/language.googleapis.com). 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)

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

# Load data
load("./data/data_tweets.RData")
```

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

```
data_tweets <-
  bind_cols(data_tweets,
            text_en = gl_translate(data_tweets$text,
             target ="en")$translatedText)
```

```
## 2021-06-17 12:44:49 -- Translating text: 13148 characters -
```

```
# Call the API via the function 'gl_nlp()' and specify your
  quantity of interest (here: analyzeSyntax)
data_syntax <- gl_nlp(data_tweets$text_en, 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
data_tweets$syntax_tokens <- data_syntax[["tokens"]]

# If you no longer need your original data (or put
  differently, if you are only interested in the single
  tokens and their respective tag), consider this
  alternative:
# Bind class of words with respective words in sentence

#syntax_short_df <- cbind(data_syntax[["tokens"]][[1]]
  [["tag"]],
                #data_syntax[["tokens"]][[1]][["value"]])

# Transform to a dataframe
#syntax_short_df <- as.data.frame(nlp_df)
```

```
# Now we are only interested in tokens that are marked as
  "nouns"
# Filter out nouns onlydata_tweets <- data_tweets %>%
  mutate(tokens_nouns = map(syntax_tokens,
                            ~ filter(., tag == "NOUN")))
# Create the data for the plotdata_plot <- data_tweets %>%
  # 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
  # delete hashtags from strings
  mutate(content = gsub("# ", "", content)) %>%
  # Unnest tokens  select(screen_name, content) %>%
  unnest_tokens(output = word, input = content) %>% #
  generate a wordcloud
  anti_join(stop_words) %>%  group_by(screen_name) %>%
  dplyr::count(word) %>%  filter(word!="https") %>%
  filter(word!="t.co") %>%
  arrange(desc(n), screen_name) %>%  filter(n > 3)
# Visualize in a word clouddata_plot %>%
  ggplot(aes(label = word,              size = n,
             color = screen_name)) +
  geom_text_wordcloud() +  scale_size_area(max_size = 20) +
  theme_minimal()
```

Figure 2.1: Wordcloud of nouns found within tweets

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.