Images are an increasingly used data source in the social sciences. One application is to extract features from human faces using machine learning algorithms. This blog post provides a guide on using APIs for this task, specifically how to access the services offered by Face++ and the Microsoft Face API. The post walks you through (1) how to gain API access credentials, (2) how to call the Face++ API from R, and (3) how to handle the output. It is based on the talk by Theresa Küntzler, who introduced the participants of the MZES Social Science Data Lab on May 12, 2020, to Extracting Emotions (and more) from Faces with Face++ and Microsoft Azure.

**Overview**

## Retrieving your API Key and Secret

* 1. Face++ API Key and Secret
  2. Azure API Key

## R Functions

* 1. Function Flow in Pseudocode
  2. The Authentification Function
  3. API Call Function

## Making the Call

1. **Choosing between Face++ and Azure?**
   1. Ethical Considerations

## Further readings

Emotions impact information processing (e.g. Marcus, Neuman, and MacKuen 2000; Meffert et al. 2006; Soroka and McAdams 2015; Fraser et al. 2012), attitudes (e.g. Brader 2005; Lerner and Keltner 2000; Marcus 2000), and decision making in general (e.g. Clore, Gasper, and Garvin 2001; Pittig et al. 2014; Slovic et al. 2007). One way of measuring the emotions of a person is by looking at their facial expressions. A new development in this area is its automation, called facial expression recognition (FER). This technique applies machine learning and deep neural networks to the classification of facial expressions in images and videos.

Commercial providers give easy access to such tools. In this post, I explain step-by-step how to make use of the provider Face++. In a very similar fashion, you can also deploy the service of Mircosoft Azure.

# Retrieving your API Key and Secret

To use the services of an API, every user needs to obtain their own access credentials. These credentials are similar to a unique username and a password. For APIs, these are called **API Key** and **API Secret**. You should keep these safely, like any other username and password.

**Face++ API Key and API Secret**

To create your API Key and API Secret for Face++, you need to first register with Face++. After logging in with your data, go to the *Console*, choose *Apps*, and *API Key*, and select *Get API Key*. A form opens, which you need to fill out. As *API Key Type*, you can choose *Free* to use the free of cost services. After confirming the form, you should be able to see your personal API Key and API Secret.

**Azure API Key**

The process for Azure is very similar, although you only receive an API Key. Again, first register with Azure. Once this is done, you see your API Key under *Portal* and the *Key*-icon.

# R Functions

In the following, I first show the general flow of the main function with pseudo code. A pseudo code describes the steps of the algorithm in plain language. After that, the actual code is explained in the example of Face++. Accessing Azure works very similarly. The full R-Scripts for API calls to both Face++ and Azure are mentioned below:

Azure\_Function\_Mzes.R

|  |
| --- |
| ## Load Packages |
|  | library(data.table) |
|  | library(jsonlite) |
|  | library(httr) |
|  | library(dplyr) |
|  |  |
|  | ## Function gathers recognition estimates from Azure |
|  | ## input: |
|  | ## 1. data: vector of fullpaths to images |
|  | ## 2. your personal key |
|  | ## output: |
|  | ## 1. dataframe with variables, listed in faces <- data.table(....); |
|  | azure <- function(fullpath, key) { |
|  | ## Initilize Object to store API output for single image |
|  | face <- NULL |
|  |  |
|  | ## create data.table with spots for all information |
|  | faces <- data.table(faceid = as.character(NA), |
|  |  |
|  | face\_rectangle\_top = as.numeric(NA), face\_rectangle\_left = as.numeric(NA), |
|  | face\_rectangel\_width = as.numeric(NA), face\_rectangle\_height = as.numeric(NA), |
|  |  |
|  | emo\_anger = as.numeric(NA), emo\_contempt = as.numeric(NA), |
|  | emo\_disgust = as.numeric(NA), emo\_fear = as.numeric(NA), |
|  | emo\_hapiness = as.numeric(NA), emo\_neutral = as.numeric(NA), |
|  | emo\_sadness = as.numeric(NA), emo\_surprise = as.numeric(NA), |
|  |  |
|  | gender = as.character(NA), |
|  |  |
|  | fullpath = fullpath, |
|  |  |
|  | error\_code = as.character(NA), error\_message = as.character(NA)) |
|  |  |
|  | ## run counts the number of image at testing |
|  | ## go over each fullpath and send to API |
|  | ## write API-output in face |
|  | run <- 0 |
|  | for (i in 1:length(fullpath)) { |
|  | run <- run + 1 |
|  | cat(run, "\n") |
|  | while(is.null(face)) { |
|  | try( |
|  | face <- as.character(httr::POST(url = "https://westeurope.api.cognitive.microsoft.com/face/v1.0/detect", # note: url changes depending on your location in the world |
|  | config = add\_headers(.headers = c("Ocp-Apim-Subscription-Key" = key)), |
|  | query = list(returnFaceAttributes = "emotion,gender"), |
|  | accept\_json(), |
|  | body = upload\_file(fullpath[i], "application/octet-stream"), |
|  | encode = "multipart")), |
|  | silent = FALSE |
|  | ) |
|  | } |
|  |  |
|  | ## if error |
|  | if (is.null(fromJSON(face)$error)) { |
|  | ## if face is found, extract information and write into data.table |
|  | facecount <- length(fromJSON(face)$faceId) |
|  |  |
|  | if (facecount != 0) { |
|  | faceid <- fromJSON(face)$faceId |
|  | face\_rectangle <- fromJSON(face)$faceRectangle |
|  |  |
|  | emotion <- fromJSON(face)$faceAttributes$emotion |
|  | gender <- fromJSON(face)$faceAttributes$gender |
|  |  |
|  | ## write info to data.table |
|  | faces[faces$fullpath == fullpath[i],][,1:14] <- c(faceid[1], face\_rectangle[1,], emotion[1,], gender[1]) |
|  |  |
|  | if (facecount > 1) { |
|  | faces <- union(x = faces, |
|  | y = data.table(faceid = faceid, |
|  | face\_rectangle\_top = face\_rectangle[,1], |
|  | face\_rectangle\_left = face\_rectangle[,2], |
|  | face\_rectangel\_width = face\_rectangle[,3], |
|  | face\_rectangle\_height = face\_rectangle[,4], |
|  |  |
|  | emo\_anger = emotion[,1], |
|  | emo\_contempt = emotion[,2], |
|  | emo\_disgust = emotion[,3], |
|  | emo\_fear = emotion[,4], |
|  | emo\_hapiness = emotion[,5], |
|  | emo\_neutral = emotion[,6], |
|  | emo\_sadness = emotion[,7], |
|  | emo\_surprise = emotion[,8], |
|  |  |
|  | gender = gender, |
|  |  |
|  | fullpath = fullpath[i])) |
|  | } |
|  |  |
|  | face <- NULL |
|  | Sys.sleep(3) |
|  | } else { |
|  | face <- NULL |
|  | Sys.sleep(3) |
|  | } |
|  | } #close if(error) |
|  | else{ |
|  | faces[faces$fullpath == fullpath[i]][,16] <- fromJSON(face)$error$code |
|  | faces[faces$fullpath == fullpath[i]][,17] <- fromJSON(face)$error$message |
|  | face <- NULL |
|  | Sys.sleep(3) |
|  | } |
|  | } |
|  | return(faces) |
|  | } |
|  |  |
|  | ## Add your Azure key |
|  | mykey = "[your key]" |
|  |  |
|  | ## Create your vector with filepaths |
|  | mypaths <- "[your vector with filepaths to images]" |
|  |  |
|  | ## call the api |
|  | faces <- azure(fullpath = mypaths, key = mykey) |

Facepp\_Function\_Mzes.R

|  |
| --- |
| ## Load Packages |
|  | library(data.table) |
|  | library(jsonlite) |
|  | library(httr) |
|  | library(dplyr) |
|  |  |
|  | ## Function creates object with Face++ key and secret, pass object to faceEst function |
|  | ## Input: |
|  | ## 1. api\_key: character, given from faceplusplus account |
|  | ## 2. api\_secret: character, given from faceplusplus account |
|  | ## Output: |
|  | ## 1. authentifiaction object to be used in faceEST function |
|  | ## Note: Function written by Sascha Goebel |
|  | authFacepp <- function(api\_key, api\_secret){ |
|  | auth <- structure(list(api\_key = api\_key, api\_secret = api\_secret), class="FaceppProxy") |
|  | } |
|  |  |
|  | ## Function gathers recognition estimates from Face++ |
|  | ## input: |
|  | ## 1. data: vector of fullpaths to images |
|  | ## 2. auth with api info (from function authFacepp) |
|  | ## output: |
|  | ## 1. dataframe with variables, listed in faces <- data.table(....); |
|  | ## Note: Original function written by Sascha Goebel; adjsted by Theresa Kuentzler to: |
|  | ## a) upload images from local machine instead of via link |
|  | ## b) save multiple faces per image |
|  | ## c) Adjustment to some changes in the API |
|  | facepp <- function(fullpath, auth) { |
|  | ## Initilize Object to store API output for single image |
|  | face <- NULL |
|  |  |
|  | ## create empty table to fill with API output |
|  | faces <- data.table(emo\_anger = as.numeric(NA), emo\_disgust = as.numeric(NA), |
|  | emo\_fear = as.numeric(NA), emo\_happiness = as.numeric(NA), |
|  | emo\_neutral = as.numeric(NA), emo\_sadness = as.numeric(NA), |
|  | emo\_surprise = as.numeric(NA), |
|  |  |
|  | gender = as.character(NA), |
|  |  |
|  | facecount = as.numeric(NA), |
|  |  |
|  | fullpath = fullpath) |
|  |  |
|  | ## run counts the number of image at testing |
|  | ## go over each fullpath and send to API |
|  | ## write API-output in face |
|  | run <- 0 |
|  | for (i in 1:length(fullpath)) { |
|  | run <- run + 1 |
|  | cat(run, "\n") |
|  | while(is.null(face)) { |
|  | try( |
|  | face <- as.character(httr::RETRY("POST", "https://api-us.faceplusplus.com/facepp/v3/detect", |
|  | body = list(api\_key = auth$api\_key, |
|  | api\_secret = auth$api\_secret, |
|  | image\_file = upload\_file(fullpath[i]), |
|  | return\_landmark = 0, |
|  | return\_attributes = "emotion,gender"), |
|  | times = 2, |
|  | encode = "multipart")), |
|  | silent = FALSE |
|  | ) |
|  | } |
|  |  |
|  | ## if face is found, extract information and write into data.table |
|  | facecount <- length(fromJSON(face)$faces$face\_token) |
|  |  |
|  | if (facecount != 0) { |
|  | emotion <- fromJSON(face)$faces$attributes$emotion |
|  | gender <- fromJSON(face)$faces$attributes$gender |
|  |  |
|  | ## write info to data.table |
|  | faces[faces$fullpath == fullpath[i],][,1:9] <- c(emotion[1,], gender[1,], facecount) |
|  |  |
|  | ## if more than one face found in image, make df with all info and merge |
|  | if(facecount > 1) { |
|  | faces <- dplyr::union(x = faces, |
|  | y = data.table(emo\_anger = emotion[,1], |
|  | emo\_disgust = emotion[,2], |
|  | emo\_fear = emotion[,3], |
|  | emo\_happiness = emotion[,4], |
|  | emo\_neutral = emotion[,5], |
|  | emo\_sadness = emotion[,6], |
|  | emo\_surprise = emotion[,7], |
|  |  |
|  | gender = gender[,1], |
|  |  |
|  | facecount = facecount, |
|  |  |
|  | fullpath = fullpath[i])) |
|  | } |
|  |  |
|  | face <- NULL |
|  | Sys.sleep(2) |
|  | } else { |
|  | face <- NULL |
|  | Sys.sleep(2) |
|  | } |
|  | } |
|  | return(faces) |
|  | } |
|  |  |
|  | ## Fill in your details |
|  | myauth <- authFacepp(api\_key = "[your key]", api\_secret = "[your secret]") |
|  |  |
|  | ## Create your vector with filepaths |
|  | mypaths <- "[your vector with filepaths to images]" |
|  |  |
|  | ## Call the function |
|  | faces <- facepp(fullpath = mypaths, auth = myauth) |

**Function Flow in Pseudo Code**

The call function takes two inputs: (1) You have to provide a vector that contains the filepath to the images on your computer and (2) the Authentification object. When calling Face++, a simple function you find below combines your personal API Key and API Secret into one object to put here. When calling Azure, you simply use your API Secret directly. With the vector of file paths a table is created with as many rows as there are files. The vector is one variable. For each variable that should be extracted from each image, an additional empty variable is created. This data table is called faces. Next, a for-loop is started that runs for every image in filepath: In the first step, call the API with the respective image. if a face is found in the image by the API, the information is written into the faces table to the line of the respective filepath. One image may contain multiple faces. if more than one face is found in the image, lines for the different faces are added to the faces table, and the information is stored. In the case of a *Free* account, the amount of calls is limited per minute. Thus, after each call, the function pauses for two seconds. Then, the call is executed for the next image. When all calls are successful, the function returns the faces data table, which is now filled with all information.

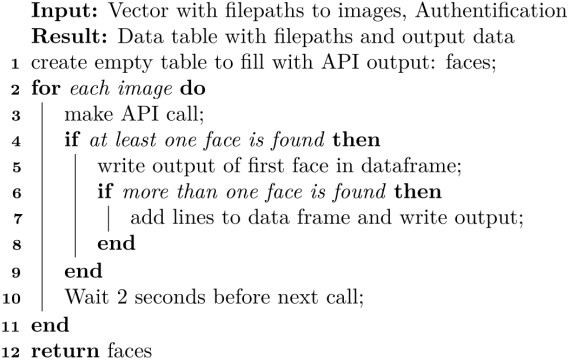


Figure 1: Pseudo code to showcase how the algorithm works

**The Authentification Function**

In a first step, you need to load the necessary packages:

library(data.table) # Extension of `data.frame` library(jsonlite) # A Robust, High Performance JSON Parser and Generator for R

library(httr) # Tools for Working with URLs and HTTP library(dplyr) # A Grammar of Data Manipulation

When calling Face++, use this authentification function authFacepp to merge the API Key and API Secret into one object.

authFacepp <- function(api\_key, api\_secret) { auth <-

structure(list(api\_key = api\_key, api\_secret = api\_secret), class = "FaceppProxy")

}

## Note: Function written by Sascha Goebel

**API Call Function**

The function that makes the call is named facepp. In the following section, the function is explained bit by bit. The first step matches the input and line 1 in the pseudocode. The function takes the two inputs fullpath, the vector with the file paths, and auth, the authentification object. The faces table is generated, with empty variables for all emotions, two additional variables extracted from Face++, and the vector fullpath. The object face is initialized. This will be used to store the information of a single image before adding it to the faces table.

facepp <- function(fullpath, auth) {

## Initialize Object to store API output for single image face <- NULL

## create empty table to fill with API output faces <-

data.table(

emo\_anger = as.numeric(NA), emo\_disgust = as.numeric(NA), emo\_fear = as.numeric(NA), emo\_happiness = as.numeric(NA), emo\_neutral = as.numeric(NA), emo\_sadness = as.numeric(NA), emo\_surprise = as.numeric(NA), gender = as.character(NA), facecount = as.numeric(NA), fullpath = fullpath

)

In the next bit of code, lines 2 and 3 of the pseudo code are implemented. A for loop is started, which runs over every element of fullpath. The run counts the number of images that are sent. It then prints the count to the console during the call. This information is added for the user to know the progress of the call. The RETRY-function executes the actual call. It states the

“Request Method” as “Post”, thus verb = "POST", it gives the “Request URL” and the “Request Parameter”, which are specified in the body of the function. The manual specifies required (api\_key, api\_secret, and image\_file) and optional request parameters. In the return\_attributes-element, it is specified which variables you would like to obtain from the API. Note that there is no space after the comma in the return\_attributes. The result of the call for a single image is stored in face.

run <- 0 #running counter of images sent for (i in 1:length(fullpath)) {

run <- run + 1 cat(run, "\n")

while (is.null(face)) { try(face <- as.character(

httr::RETRY( verb = "POST",

body = list(

api\_key = auth$api\_key, api\_secret = auth$api\_secret,

image\_file = upload\_file(fullpath[i]), return\_landmark = 0,

return\_attributes = "emotion,gender"

),

times = 2,

encode = "multipart"

)

),

silent = FALSE)

}

To better understand what the API output for a single image looks like, lets print one. The output is a character vector in the JSON format. You can spot the information that is added, such as a number given to each emotion value or the predicted gender. JSON files follow specific rules, making it easy to extract the information and write it into the faces table. In R, this can be done with the fromJSON function from the jsonlite package.

print(face)

## [1] "{\"request\_id\":\"1600002798,3c7cc10b-fe59-4fa5-9a1a- e2375a35ceee\",\"time\_used\":146,\"faces\":[{\"face\_token\":\" 2a46a4591795444441bde5e159469b2a\",\"face\_rectangle\":{\"top\":146,

\"left\":300,\"width\":195,\"height\":195},\"attributes\":{\"gender

\":{\"value\":\"Male\"},\"emotion\":{\"anger\":0.000,\"disgust\": 0.000,\"fear\":0.063,\"happiness\":99.937,\"neutral\":0.000,\"sadness

\":0.000,\"surprise\":0.000}}}],\"image\_id\":\"KdgNW2IvGLbViBZ1ialuLQ=

=\",\"face\_num\":1}\n"

Before continuing, it is necessary to check whether a face was found by the API (line 4 in the pseudocode). To so, the number of face\_tokens is counted. Face++ assigns a unique identifier to each face per image, called face\_token. if at least one face token is found, the information is extracted. The values for all emotion variables, for all faces identified in an image, are extracted into the emotion object. The same is done for the gender values into the gender

object. Next, the information for the first face is used to fill the empty line in the faces table (line 5 pseudo code). if more than one face is found in the image, the information for all faces found is stored in a data.table. This newly created table is unioned with the faces table. Thereby, all lines that were not in the faces table before being added. The line for the first face of the image is not added since it already exists in faces (lines 6 and 7 in the pseudo code).

## if face is found, extract information and write into data.table facecount <- length(fromJSON(face)$faces$face\_token)

if (facecount != 0) {

emotion <- fromJSON(face)$faces$attributes$emotion gender <- fromJSON(face)$faces$attributes$gender

## write info of first face to data.table faces[faces$fullpath == fullpath[i],][, 1:9] <-

c(emotion[1,], gender[1,], facecount)

## if more than one face found in image, make df with all info and merge

if (facecount > 1) { faces <- dplyr::union(

x = faces,

y = data.table(

emo\_anger = emotion[, 1], emo\_disgust = emotion[, 2], emo\_fear = emotion[, 3], emo\_happiness = emotion[, 4], emo\_neutral = emotion[, 5], emo\_sadness = emotion[, 6], emo\_surprise = emotion[, 7], gender = gender[, 1], facecount = facecount, fullpath = fullpath[i]

)

)

}

The end of the facepp contains some housekeeping (lines 10 to 12 in pseudocode): The information in the face object is deleted to avoid overlap with the information from the next image. Whether a face was found or not in the previous image, the function pauses for two seconds. Once all images from the fullpath vector have been sent to the API, the function returns the faces table that contains all information now.

face <- NULL Sys.sleep(2)

} else {

#if no face was found face <- NULL Sys.sleep(2)

}

}

return(faces)

}

# Making the Call

To finally make the call, run the two functions so that they are part of your environment. Then use your API Key and API Secret to call the functions:

## Fill in your details myauth <-

authFacepp(api\_key = "[your key]", api\_secret = "[your secret]")

## Create your vector with filepaths

mypaths <- "[your vector with filepaths to images]"

## Call the function

faces <- facepp(fullpath = mypaths, auth = myauth)

# Choosing between Face++ and Azure?

This article presents two alternative tools for the same task. Both tools have different strengths and weaknesses. In the best case, some of the specific data to be classified as ground truth, for example, from manual coding. To make an informed choice, which of the tools performs better on the specific data, my suggestion would be to try a sample with both tools and compare.

Colleagues and I test and compare Face++, Azure, FaceReader .One of our findings is that all tools perform very well on prototypical facial expressions under good conditions (such as good lighting and frontal camera angle). Errors occur more frequently when the facial expressions are more subtle, or images taken in darker environments. In addition, while Face++ performs much better in face detection (so it is better in ‘finding’ faces in images), Azure does not recognize as many faces. However, Azure shows a better categorization of the emotions in the sample used for our analyses once a face is found. In sum, choosing the tool that promises the most reliable outcome is an individual decision depending on the data to be classified.

**Ethical Considerations**

When making use of third-party services, I advise to consider additional ethical and data security issues.

First, one should be aware of the fact both Azure and Face++ are offered by for-profit companies, and by using them, one supports their efforts. Problems that can arise here became heavily evident when Megvii Technologies, the company that develops Face++, was associated with China’s mass surveillance system IJOP in Xinjiang.

Second, I wish to highlight that the companies can use the images sent to the API according to their Terms of Service, such as further processing of the user data or potentially using the

images sent to the API for internal research. Please read and understand the Terms of Service before using any such tools.

Third, in addition to general performance considerations, artificial intelligence products are currently developed with strong biases with respect to culture, race, gender, and more .