The use of cloud for generated statistics data file storage in statistics tasks

Mikuláš Gangur

Department of Economics and Quantitative Methods, Faculty of Economics, University of West Bohemia, Univerzitní 8, Plzeň, Czech Republic

gangur@kem.zcu.cz

Abstract

The paper builds on previous work that dealt with the automatic generator of parameterized tasks and tests. In these works, the author focused mainly on the issue of automatic generation of parameterized tasks in the field of data science. A key problem in creating tasks of this type is the generation of source data and especially their storage and subsequent access in various systems and formats. This contribution is an innovation of previously used procedures and an extension of the possibilities of working with synthetic data files. The innovative data storage system uses cloud storage for its work and thus simplifies the work of the generator user when generating tasks that also contain the statistics data stored in the data file. No knowledge of cloud technologies is required to generate these tasks. In this solution we can work with the statistics tasks containing data files not only in format of selected LMS (for example LMS Moodle), but we can work with these tasks published in PDF format, where the data is represented by a link to the cloud storage. The contribution is more technical and contains the procedures and codes needed to work with cloud storage in both directions – data storage and data retrieval. These procedures also include instructions that allow you to create and use cloud storage. An integral part of the solution is also the design of the administration system of stored data and their periodic cleaning.

Keywords

Generator of parameterized tasks. Generation of random data collection. Statistical data file. Cloud. Matlab.

Introduction

Automatic generation of parameterized task can generate a unique problem for each student. Students can practise one task with different input parameters. It helps them to understand the problem from different points of view. In the article (Gangur, 2011) the basic principles of an automatic generator of questions and answers are described. The generator of algorithms generates random input parameters of a question according to problem constraints. At the same time, the generator solver calculates and generates responding answers based on the input parameters according to the problem solution. The outputs of the procedure consist in creating questions meant to be imported to a question bank and/or in creating a set of randomly selected and automatically created questions for the whole test. Both outputs can be generated within the demanded structure suitable for the applied system. The output of the generating process is a description of tasks in universal XML format. From this format, tasks can be converted to the desired target format using the XSLT process. One of the generator target output formats can be for example Moodle XML which is used to import data into LMS Moodle Question bank.

A specific approach is applied when generating tasks from the field of statistics and statistical processing as well as an analysis of the data. In this case just the statistical data are the key object. Creating the data according to the required parameters, storing the data in the file and inserting the information about the file must be an integrated part of the generation process, as well as the information about the data themselves and then storing them in the output description of the generated task. In (Gangur, 2018) automatically generated data files are attached as a part of a statistical task. This feature of the tasks followed previously developed generator feature, i.e. automatic generation of parameterized images (Gangur, 2016), (Plevný, & Gangur, 2016).

In (Gangur, 2018) the problem of statistical data files storage on cloud was introduced together with an explanation of the advantages of generating synthetic statistical data as an integral part of a given task. Two types of statistical data were used. In the first group there are data files that are closely linked with the given generated task. In case of generating a task into the tasks bank in LMS these files are part of a file with the description of the entire task. For example, in statistical tasks these are relatively smaller selective files with the data selections which are unique for each task.

In the second group there are data files that can be joint for more tasks. Such files often contain a vast volume of data and it is not desirable to store them in repository together with the definition of each task. Basic files with a vast volume of the whole population may be an example of that. This type of statistical data storage has gradually become universal and links to this online data on the network is also part of statistical tasks in question bank.

This new approach and the large increase in cloud storage files has required more flexible administration in storing and cleaning these data files in the cloud. This paper shows the solution to this problem as it is currently used by the automatic task generator.

The reminder is structured as following. Next section provides an overview of the literature on the problem of automatic generation of synthetic statistical data. Following section describes the process of data files storage on cloud and administration of files on cloud. Finally, the advantages and limitations of proposed solution are discussed, and the conclusion is stated. In the following parts, the methods are presented on an example of Matlab functions (MathWorks, 2013). Unlike other development environments, the presented relatively simple solutions developed and tested by the author, are not at the very least, directly available and easily searchable.

RELATED WORKS

Many resources offer statically created datasets that are used in teaching statistics. For all can be mentioned for example (Freeman, 2021) or (State, 2021). Such data can be used as static or it is possible to generate various selections with the required properties from these datasets.

The generated data, also called synthetic data, is offered on some websites created using built-in generators. They are used by teachers to teach their stats class. They know data for a statistical test, and they know the result. As example two websites exist that do just that. One tried, and trustworthy resource was created by I/O psychologist Richard Landers (Landers, 2021). Second one social psychologist Andrew Luttrell offers (Luttrell, 2021). Both allow interactive user-friendly generation of statistical data files for further use in teaching statistics. However, none of such web sites or other similar sources offer APIs for flexibly obtaining generated datasets according to specified parameters and using them further to automatically generate statistical tasks. Another example of the use of prepared non-interactive synthetic data in teaching can be found in (Mobahi & Min, & Wojtusiak, 2019).

Another approach is used by the author in (Arifin, 2011). The process of creating a dataset is made with the utilization of PASW Statistics to generate random values. The objective of the contribution is to demonstrate the creation of data which are measured on continuous scale, using PASW Statistics menus and syntax. Even in this case of using the SW tools, the generated data are used for teaching, but dynamic connection to other systems creating corresponding statistical tasks over the generated data is not applied.

Compared to the above examples of the use of generated data sets, the automatic generator system (Gangur, 2018) generates synthetic data as part of a statistical task, and the structure of the generated data interacts with the assignment of the generated task. The problem with the original version of the generator was in the administration of the generated data files, i.e. their storage. This process was inflexible and "manual" rather than automatic. This contribution seeks to overcome this shortcoming by utilizing the cloud and the Google Service Drive API. In the following the article explains the design of automatic upload process of generated data files to the cloud using Google Service Drive API.

The described solution thus enables the generator user (teacher, task author) to conveniently create statistical tasks and especially statistical data for the given tasks at the same time and to create a dynamic connection between them. Automatically generated statistical data are thus an integral part of a given statistical task. This approach allows the use of statistical tasks with data not only in the repository of the task used by the LMS but is especially suitable in cases of generating tests in PDF format, in which dynamic links to data files are created.

Methodology

Our system uses the Google Service Drive API for automatic storage of files. The whole process we can describe in four main steps.

1. Generation of access token in Google drive.
2. Save data to temporary file on local disk. This step is for data controlling.
3. Upload the content of local temporary file to Google disk drive under demanded file name.
   1. Refreshing of access token to Google drive.
   2. Uploading data from local temporary file to Google drive disk under demanded name (see Listing 1).
4. Downloading the data file from cloud under demanded file name (see Listing 4).
5. Cleaning files on drive.

Procedure for generating an access token

First, the access token to your account on cloud must be generated. This token is required for any access to cloud via Google Service Drive API. The token can be received as result of project registration in Google Service Drive API (Google, 2021a) and (Google, 2021b). The created token is saved in file in working directory. In Matlab system the token is in system mat file, for using by PHP the token and client secrets are saved in json files.

The process of saving data to local temporary file

In this step generated statistical data in *dataTask* variable are saved to local temporary file under name *dataFileName*. The data are saved with help of different Matlab functions with respect to format of input data.

The process of uploading file data to Google drive

In process of uploading data file the main function *UploadExcelToGoogleDrive()* called from task generator uploads data in local file *dataTmpFile* to cloud under name *nameFile* and sets the file life on cloud. The function *UploadExcel ToGoogleDrive()* (see Listing 1) refreshes access token to Google Service Drive for next using and then it retrieves access token for current uploading (see Listing 2). Then the function *uploadFileToGoogle() is* called as the key part of file uploading.

Listing 1: Refreshing access token and uploading file to cloud (Source: own)

% Upload coded xlsx file in base64 as media, i.e. upload of one % file under demanded name on cloud.

function output = UploadExcelToGoogleDrive(dataTmpName, ...

nameFile, ...

lifTime)

% Refresh and retrieve access token

aDrive = refreshAccessToken();

...

% Uploading file to Google drive

output = uploadFileToGoogle(aDrive, dataTmpFile, ...

nameFile, lifeTime);

end

The refreshing token process load the *client\_id* and *client\_secret* from mat file. In second step the process connects to Google Service Drive API accounts and the token is refreshed with help of authentication information. New access token is returned, and all new information are saved again to mat file.

Listing 2: Refreshing and retrieving access token (Source: own)

function aDrive = refreshAccessToken()

...

load google\_tokens.mat

options = weboptions('RequestMethod','POST');

newAccessTokenString=webread('https://accounts.google.com/o/oauth2/token','client\_id', client\_id, 'client\_secret',client\_secret, 'refresh\_token', rDrive, 'grant\_type', 'refresh\_token',options);

aDrive=newAccessTokenString.access\_token;

save('google\_tokens.mat', 'aDrive', '-append');

end

Function *uploadFileToGoogle()* play the key role in process of uploading statistical data to cloud (see Listing 3). The uploading process consists of two steps. In first one the file is uploaded to Google drive and in the second one the permissions are set.

The function uses the classes http.io.MultipartFormProvider, http.io.JSONProvider, http.HeaderField and http.RequestMessage. This method creates object for request creation and its sending to Google Service Drive API access point. The request consists of two parts – message header and message body. The object http.HeaderField represents the message header and following parameters are included in header. The method of authorization is set via AccessToken and Content-Type is marked as multipart data.

* 'Authorization', 'Bearer '= AccessToken
* 'Content-Type'='multipart/form-data'

More parts in message body are made up of metadata and own data. Metadata described the characteristics of data using the following parameters:

* metadata.name = nameFile;
* metadata.mimeType = 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet';
* metadata.properties.lifeTime = lifeTime;
* metadata.parents = idFolder;

The parameters define the name of data file on cloud, the id of parent directory, where the data files are stored and very important information about type of statistical data. This parameter limits the format of transferred data only to spreadsheet data. This type is enough to solve the problem of statistical data transfer. Important property is lifeTime in added properties. This property together with built-in file property cretatedTime allow later administration of files on cloud according their expiration. The metadata are transformed to JSON format using the object JSONProvider.

The statistical data are second part of message body. The data are red in corresponding character set with respect to DefaultCharacterSet value. Different character sets can be selected (UTF-8, windows-1250 ect.).

Finally, the object MultipartFormProvider creates the message body from metadata and statistical data and the object RequestMessages finishes the message creation from message header and message body. The process of file uploading ends with setup of Google Service Drive API access point url and with sending request using send method of RequestMessage object. The url also includes parameters uploadType=multipart and supportsAllDrives=True that enable to send multipart message body and upload the file to all type of drives, including shared drives.

In the second step the permissions to file are set. The request is sent with different service access point according to id of file received from previous upload request and with different parameters. The header includes following parameters, because the request consists in body only one part in JSON format:

* 'Authorization', 'Bearer '= AccessToken
* 'Content-Type'='application/json'; charset = UTF-8
* 'Accept' = 'application/json'

The permission setting is the most important part of this process. The permission role must be set to *reader* and permission type to *anyone*. Then the permission is transferred to JSON format as only one part of message body. The construction of request from header and body is realized by the same way as in first step of uploading with the help of RequestMessage object. The Google Service Drive API access point unlike the uploading request must contain file id on cloud that is received from response of uploading request in first step. The name of service is also different. In the first step the upload and in the second one the permission is used.

Listing 3: The key part of statistical data uploading (Source: own)

function output=uploadFileToGoogle(AccessToken, dataTmpFile, ...

nameFile, lifeTime)

import matlab.net.http.io.MultipartFormProvider;

import matlab.net.http.io.JSONProvider;

import matlab.net.http.HeaderField;

import matlab.net.http.RequestMessage;

% The header of message

HeaderField = HeaderField(

'Authorization',['Bearer ', AccessToken], ...

'Content-Type','multipart/form-data' ...

);

% Set id directory on cloud for uploading of files.

idFolder{1} = '0AJCqk3je7B8AUk9PVA';

% Metadata describe the properties of file on cloud in JSON.

metadata.name = nameFile;

metadata.mimeType = 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet';

metadata.properties.lifeTime = lifeTime;

metadata.parents = idFolder;

metadata = JSONProvider(metadata);

code = feature('DefaultCharacterSet');

feature('DefaultCharacterSet','windows-1250');

% Read the file content

data = fileread(dataFile);

formProvider = MultipartFormProvider('metadata',metadata, ...

'data',data);

% The whole message creation

req = RequestMessage('POST',HeaderField,formProvider);

% The creation URL of Google Service Drive API access point

url = 'https://www.googleapis.com/upload/drive/v3/files?

uploadType=multipart&supportsAllDrives=True';

% Send message to Google drive service

response = req.send(url);

feature('DefaultCharacterSet',code);

...

% ===========================================================

% Setting of authorization for file sharing in JSON

Permission.role='reader';

Permission.type='anyone';

bodyPerm = JSONProvider(Permission);

% The creation of message header

HeaderFieldPerm = matlab.net.http.HeaderField

('Authorization',['Bearer ', AccessToken], ...

'Content-Type','application/json; charset=UTF-8', ...   
 'Accept','application/json' ...

);

reqPermission = RequestMessage('POST',HeaderFieldPerm,bodyPerm);

% The creation URL of Google Service Drive API access point. The % id from response of previous request is included

url = sprintf('https://www.googleapis.com/drive/v3/files/%s/

permissions?&supportsAllDrives=True',response.Body.Data.id);

% Send message to Google drive service

reqPermission.send(url);

...

end

The process of downloading file data from cloud

The data file was stored under demanded name, but the used url can’t work with this demanded name, because the file is stored on cloud under unique id. The script must be applied for download the data file from cloud to local user’s disk. The automatically generated link to statistical data file in created statistical tasks can have following format for PHP script language:

https://<public directory>/FindFile.php?nameFile=<demanded name>

This link is generated as part of statistical tasks and it is included in final PDF file as a working http link.

The script for file downloading can be developed in any suitable language. The important part of working with script is management of access tokens. The files with access token and client\_secret, that we received during the Google Service Drive project registration, must be prepared in JSON format in directory of PHP script before downloading script using. In scripts we use classes Google\_Service\_Drive and Google\_Client from GitHub (GitHub, 2021). Only one input parameter of script is demanded name of data file nameFile as one of metadata items of file data on cloud (see Listing 4). The functions GetToken() and ReturnFileID() are the key parts of downloading script. The accessToken allows us to get the ID of data file on cloud according to nameFile and idFolder. Retrieving a file and its ID uses a query like retrieving a list of files to clean (see Listing 5). Received ID allows us to construct request url of Google Service Drive access point for downloading data file with this ID. Finally, we start downloading process with help of header function. Similarly, as ID, we get specified properties of files in later administration of files on cloud.

Listing 4: Main part of downloading process in PHP (Source: own)

...

$nameFile = strip\_tags(htmlspecialchars($\_GET["nameFile"]));

$accessToken = GetToken();

$idFolder = '0AJCqk3je7B8AUk9PVA';

$idFile =ReturnFileID($idFolder,$nameFile,$accessToken);

...

$url ='https://drive.google.com/file/d/'.$idFile.'/view?

usp=sharing';

...

header("Location: ".$url);

The function GetToken() performs two main tasks similar to access in uploading process – to get token from file token.json and to refresh the token with help of information client\_id and client\_secret in client\_secret.json file. The refresh token is moved to access token and the refresh token is updated. The information is saved again back to file token.json for next using. All mentioned operations are performed with the help of Google\_Client object and its methods.

In ReturnFileID() the object Google\_Client is set with information from client\_secret.json file and with access\_token from token.json file. The object Google\_Service\_Drive accept the information from Google\_Client object and then its method files-> listFiles finds all files on cloud according to demanded name nameFile and idFolder as parent directory. The parameter trashed is set to false to exclude trashed files. The field ID is extracted from the result as demanded output of the ReturnFileID function.

Cleaning files on drive

Accumulating files on drive requires their administration with respect to free drive space. Expired files must be deleted automatically. The two previously mentioned properties of the createdTime file and the added lifeTIme property are used to identify such files. Their values can be used to identify expired files and then delete them. This process can be started periodically using the cron service or called in one of the described routines - file upload or download. From this point of view, it seems more convenient to clean the disk with the Matlab CleanGoogleDrive () function before uploading files to free up disk space.

Cleaning process consists of two steps (see Listing 5). In first one the all non-trashed files in drive idFolder are received. In this case, the GET request without message body is used unlike file upload and mainly the parameters q and fields are included to request URL. Question q specifies parent folder and non-trashed files. Parameter fields determines the structure of the information of each file in the output list. The final url is constructed with the help of object matlab.net.Query Parameter, then it is encoded and sent to Google Service Drive API. The expired files are selected with respect to values of createdTime, lifeTime and current time. In second step the function CleanFile accept access token and file id and it deletes the file (see Listing 6). Unlike previous requests the DELETE request is applied without message body. The id of deleted file is inserted as part of basic URL.

Listing 5: Cleaning expired files from drive (Source: own)

function output = Clean(AccessToken)

...

q = sprintf("'%s' in parents and trashed=false",idFolder{1});

fields = 'files(id,name,createdTime,properties)';

url = 'https://www.googleapis.com/drive/v3/files';

U = matlab.net.URI(url);

U.Query = matlab.net.QueryParameter('q',q, ...

'fields',fields, ...

'supportsAllDrives','True', ...

'includeItemsFromAllDrives','True');

url = U.EncodedURI;

...

Listing 6: Cleaning file from drive (Source: own)

function out = CleanFile(AccessToken,id)

...

reqFile = matlab.net.http.RequestMessage('DELETE',

HeaderFieldFiles);

url=sprintf('https://www.googleapis.com/drive/v3/files/%s',id);

U = matlab.net.URI(url);

U.Query = matlab.net.QueryParameter('supportsAllDrives','True',

'includeItemsFromAllDrives','True');

url = U.EncodedURI;

response = reqFile.send(url);

...

The cleaning process can be time consuming and therefore it is not advisable to trigger it when uploading each file in the Matlab function UploadExcelToGoogleDrive(). The periodic activation of cleaning process is therefore controlled by the time of the last activation and the set period at begin of function CleanGoogleDrive() (see Listning 7). Both values are stored in a mat file and the time of cleaning is updated each time it is run successfully. Therefore, setting the frequency of the cleaning is important step of administration process.

Listing 7: Main function for cleaning drive (Source: own)

function output = CleanGoogleDrive()

load('FrequencyCleaning','Period','LastUpdate')

if ((LastUpdate+Period) < now())

aDrive = refreshAccessToken();

...

ok = Clean(aDrive);

if (ok)

LastUpdate = now();

save('FrequencyCleaning','Period','LastUpdate')

end

end

end

conclusion

The presented functional solution offers not only automatic upload of data files to the cloud as a usable function in its own program, but also subsequent connection to these files via a flexible generated link. The generated file names in the cloud are readable, memorable and allow manual entry when downloading these data files from cloud. The described functionalities can be used to integrate the generated synthetic data and the task in the automatic generation of statistical problems, where the data are directly connected to the task via the created link. Another advantage of the system is the ability to set the lifetime of the data file, which is used when cleaning up expired data in the cloud. With a properly set lifetime, temporary data can be distinguished from long-term data files. Finally, the advantage of this solution is also the simple implementation using the built-in libraries of the Matlab system without the need for the sometimes-demanding installation of external 3rd party products packages. Implementation in the Matlab system is also a restriction of system use only for users of the system. However, the described principles can be easily applied when implemented in other development systems using the appropriate libraries.

# Reference

Arifin, W. (2011). Creating artificial data for teaching of statistics. *Vol.3*(1), pp. 69-77.

Freeman, J. (2021). Mathematics and Statistics Help (MASH). *State University of Sheffield*. Retrieved from: https://www.sheffield.ac.uk/mash/information

Gangur, M. (2011). Automatic generation of cloze questions. In: *CSEDU 2011 - Proceedings of the 3rd International Conference on Computer Supported Education*. CSEDU 2011 - Proceedings of the 3rd International Conference on Computer Supported Education, Portugalsko: SciTePress - Science and Technology Publications.

Gangur, M. (2018). Automated Generation of Statistical Tasks. In: *DIVAI 2018 - 12th International Scientific Conference on Distance Learning in Applied Informatics*. Praha: Wolters Kluwer.

Gangur, M. (2016). The Quiz Questions with Automatically Generated Images. In: *DIVAI 2016 - 11th International Scientific Conference on Distance Learning in Applied Informatics*. Praha: Wolters Kluwer.

GitHub. (2021). . Retrieved from: https://github.com/googleapis/google-api-php-client

Google. (2021a). Configure the OAuth consent screen. *Google Workspace for Developers*. Retrieved from: https://developers.google.com/workspace/guides/configure-oauth-consent

Google. (2021b). Create access credentials. *Google Workspace for Developers*. Retrieved from: https://developers.google.com/workspace/guides/create-credentials

Landers, R. (2021). Dataset Generator for Learning Introductory Statistics. *TNTLAB at University of Minnesota*. Retrieved from: https://rlanders.net/dataset-generator/

Luttrell, A. (2021). Data Generator for Teaching Statistics. *Andrew Luttrell*. Retrieved from: http://andyluttrell.com/datagen.html

MathWorks. (2013). Matlab. *MathWorks*. Retrieved from: http://www.mathworks.com/

Mobahi, H., Min, H., & Wojtusiak, J. (2019). Synthetic Data for Teaching Data Integration in Informatics Graduate Program.

Plevný, M., & Gangur, M. (2016). On the Possibility of Solving the Problem of Automatic Generation of Images in Quiz Questions. In: *DIVAI 2016 - 11th International Scientific Conference on Distance Learning in Applied Informatics*. Praha: Wolters Kluwer.

State, N. (2021). Datasets for Teaching and Learning. *NC State University Libraries*. Retrieved from: https://www.lib.ncsu.edu/teaching-and-learning-datasets