IS883 - Lab 2. Machine learning on GCP

9/20/2020

We will build experience using some of the machine learning services available on Google Cloud Platform, specifically Vision API and AutoML for image classification. The analytics assignment will build on this experience using the Vision API.

Remember to stop your instances, delete unused projects and undeploy models when you are done with them to prevent charges.

0. Clone the required repository

Open your cloud shell and clone the repository with this week's contents from Github:

```
git clone https://github.com/jgbenade31/IS883-21/
```

You should now see a directory 'IS883-21' in you cloud shell environment, containing 'lab2-class' and 'lab2-assignment'. We will work with 'lab2-class' for now.

1. Vision API

Google's Cloud vision API provides easy access to a variety of services. Try out the **demo** for yourself.

1.1 Getting set up

Open up an existing project or create a new one, and make sure billing is enabled. Check that you are authenticated and in the correct project by running:

```
gcloud auth list
gcloud config list project
```

You should see your email under the credentialed accounts, and the project id of the project you are using as `project`. If the wrong project appears here you can set it to the right one with

```
gcloud config set project [project-id]
```

Open your cloud shell (top right in the dashboard) and create a directory for week 2.

```
mkdir lab2
```

1.2 Enable APIs and create a service account

We have to enable the cloud vision and translation APIs before we can use them:

```
gcloud services enable vision.googleapis.com
gcloud services enable translate.googleapis.com
```

Client applications that use an API must be authenticated and granted access to the requested resources. Using a *service account* is the recommended way of doing this. This provides credentials for the applications, not the end-users, so the service account is owned by the project. Service accounts are associated with public/private key pairs. When you create keys, you download the private key, which is used to generate credentials when calling the API.

First, we export the project id to an environment variable, which we will refer to later.

```
export PROJECT_ID=$(gcloud config get-value core/project)
```

Now we create a new service account with access to the Vision API:

```
gcloud iam service-accounts create my-vision-sa \
  --display-name "my vision service account"
```

The next step is to create private key credentials that our Python scripts will use. These credentials are created as a JSON file `~./vision-key.json` (you should run this command as it is written, you do not need to replace PROJECT_ID with your project id since we created an environment variable to store the value) :

```
gcloud iam service-accounts keys create ~/vision-key.json \
  --iam-account my-vision-sa@${PROJECT_ID}.iam.gserviceaccount.com
```

You can check that the private key was created by checking the contents of your home directory (`ls ~`). Finally, we have to direct the project to use this private key by default. We do this by assigning the private key to the `GOOGLE_APPLICATION_CREDENTIALS` environment variable:

```
export GOOGLE_APPLICATION_CREDENTIALS=~/vision-key.json
```

Note: This will only set the variable for the current session. You will have to re-assign this environment variable everytime you log in to the cloud shell by re-running the line above. This should be your first step if you ever have see the error *'PermissionDenied: 403 Your application has authenticated using end user credentials from the Google Cloud SDK or Google Cloud Shell which are not supported by the vision.googleapis.com.'*

1.3 Upload the `detect_objects.py` script

Navigate to your `IS883-21/lab2-class` folder, where you'll find the script `detect_labels_text.py`. **Note**: If instead of cloning the repository you downloaded the file from the course page, there are a couple of ways to upload: (1) click on the three vertical dots top right of the cloud shell and select 'Upload File.' (2) click 'Open Editor' (top middle of the cloud shell), and drag the script into the right folder. (Notice that this also provides a handy in-browser editor if you don't like nano/vim/emacs.)

Once you navigated to the folder containing the script, let's run it.

```
python3 detect_objects.py
```

You should see a message that the recommended usage takes one argument, an image URI, followed by some detected objects and texts. Since we did not pass any arguments, a default public image was used, inspect the image to make sure the results make sense by visiting the given url.

1.4 Upload the `detect_translate.py` script

Navigate to your `lab2` folder, and upload the script `detect_translate.py` (if neccessary).

Let's try running it run it.

```
python3 detect_translate.py
```

You should see a message that the recommended usage takes two optional arguments:

- An image URI - the default image is the same as before
- An optional ISO 639-1 language code representing the language to which text must be translated - the default is to translate to English ('en').

1.5 Call the API on your own images

The scripts above take the location of any cloud resource as argument and calls the Vision API on it. Create a storage bucket (using the console) by navigating to [≡] > Cloud storage > Create bucket. You can use the default settings when creating the bucket, except for the following two changes: - Uncheck "Enforce public access prevention on this bucket" - Allow fine-grained access (under access control), not uniform bucket-level access.

Now copy the images under 'images/vision-api' into the bucket by clicking on your bucket name then 'Upload files'. We need make the images publicly available. For each image, use the vertical dots (far right side of the screen), select edit permissions and add a permission where *entity* is 'Public', *name* is 'allUsers' and *access* is 'Reader'.

(Note: if you have many images in your bucket it would be terribly tedious to change their permissions one at a time. An alternative is to make your entire bucket publicly accessible. To do this, set your bucket to 'Uniform' access control, go to 'Permissions' and click 'Add members'. The new member should be 'allUsers', and select 'Cloud Storage > Storage Object Viewer' in the role drop-down. Of course, making the entire bucket publicly available means you should be pretty careful about what you put in it. . .)

Now the images should be available for analysis. Try running in ithe cloud shell

```
python3 detect_objects gs://[storage-bucket]/[image-name.jpg]
```

on various images (remember to replace `[storage-bucket]/[image-name.jpg]` with the path to your bucket and file). If you see the text 'Labels (and confidence scores)', but no actual labels or confidence scores it means that the script was not able to access the image. When this happens make sure the is accessible by double-checking the steps above. Similarly, if you receive an error while running ''detect_translate.py" the most likely cause is that the image was not accessible.

1.6 Understand the API responses

Spend some time making sure you understand how the API call worked, what was returned and how you can access it. The API Explorer that is part of the documentation (for example, for character recognition and object labelling) may be useful for this.

1.7 Try calling other services

The provided script only calls the labelling, text detection and translation services, but this is jus the start of what is possible. Try to extend it to call some of the other services. You can find the available APIs as well as examples of how to call them in the documentation.

2. AutoML

When we used the Vision API we relied on previously trained models to detect and classify objects. AutoML gives you the opportunity to train your own models through a simple interface and takes care of parameter tuning and evaluation for you.

2.1 Getting set up

Download the images zip file provided on the course webpage to your local system.

Navigate to the project you want to use (or create a new one) and search for 'vision' in the search bar. Once you are on the Cloud AI Vision page, select 'Get started' next to image classification, and enable the AutoML API. This may take a couple of minutes.

Navigate to '[≡] > Cloud storage' in and create a storage bucket by using the console and default settings.

2.2 Upload training images

Navigate back to the AutoML UI, go to datasets > new dataset. Give your dataset a name and select the default option (single-label classification).

We can import training images from a storage bucket but for this exercise we will instead go through the process manually. Select 'Upload images from your computer', select the folder of images you want to upload and choose a destination on your storage bucket to save the image - I suggest creating a folder for each class of image you train on, for example `my_bucket/lion`, `my_bucket/rhino` etc. I provide images for a couple of potential classes in the images folder, but you can use whatever you like.

After completing this process for all the images you want to train on (at least two classes), you should see a notification that the images are being imported. This may take a while.

Note: At this point you may receive an error message saying that the storage bucket is inaccessible and instead you need a bucket in a specific location. If this happens just go create a new bucket in the location they provide.

2.3 Labelling images

Once all your images have been imported, go to the 'Images' tab, create labels for your classes and start labelling the images. You should have at least 10 images per label. You can also use this step to remove images that you think will confuse the training.

2.4 Training a model

Due to the potential costs involved you do not have to train you model.

When all your images are labelled, go to the 'Train' tab. You'll notice that the images are automatically split into training, validation and testing sets. The training set is used to build the models, the validation set validates the parameter choices and the testing set is used to ensure that the model generalizes to previously unseen data. I recommend leaving this unchanged.

Select 'Start training', set your node hour budget (you can use the minimum allowed - 8 hours) and select the option to deploy the model to one node after training. Note that every account gets 40 free node hours of training time, thereafter charges are applied. Refer to the pricing guide for more information (when I checked it was roughly $20/node hour, so be careful!).

With an 8 node hour budget, your model should take around one hour of wall clock time to train.

2.5 Evaluate your model

Once training has completed, take a look at the evaluation metrics for your model and upload some mew images for it to classify. The model is now available to use in whatetever application you want through a REST API, or you can download it.

Cleaning up

Remember to stop/delete/undeploy unused resources.