IS883 - Analytics assignment using Vision API

9/20/2021 - (update 9/20)

Suppose we considered building infrastructure to provide a platform that allows users to upload images for some purpose (image a marketplace like ebay, or an image-sharing website like unsplash, etc.). An integral component of this is to provide users the ability to upload new images (designs), verify that the images do not contain explicit content and automatically label images so that uploads are easily searchable. We will take a small step in this direction using the Vision API.

The goal of this assignment is to call the Vision API to (1) determine whether an image contains explicit content, and (2) label the image.

You are provided access to a web app that allows the user to upload images. Uploaded images are saved in a storage bucket and, along with their labels and classification as safe or not, entered into a Datastore.

Logistics

You may find it useful to complete Lab 2 before starting this assignment, since it provides a couple of examples of how to call the Vision API. You may also refer to the documentation **here** for guidance and inspiration.

This is an individual assignment: you may ask colleagues for help getting set up (section 1) but please do not discuss any edits to the file `main_assignment1.py`.

1. Getting set up

1.1 Housekeeping

Create a new project and make sure billing is enabled. Launch the cloud shell and 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 (replace `[your-project-id]` with the id of your project)

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

1.2 Download the sample code

Open your cloud shell (top right in the dashboard) and download the sample code from github. (If you haven't already done so as part of lab 2.)

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

You should see a folder IS883-21 in your root directory. Navigate there, then to the folder 'lab2-assignment/app'.

```
cd IS883-21/lab2-assignment/app
```

1.3 Enable APIs and create a service account

We have to enable the cloud vision, cloud storage and datastore APIs before we can use them:

```
gcloud services enable vision.googleapis.com
gcloud services enable storage-component.googleapis.com
gcloud services enable datastore.googleapis.com
```

Now we'll create a service account as in lab 2. First, export your project id to an environment variable:

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

Create a service account

```
gcloud iam service-accounts create ass1-sa \
  --display-name "assignment1 service account"
```

Set the appropriate permissions (do not replace anything, `${PROJECT_ID}` looks up the value of the environment variable `PROJECT_ID` we created):

```
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
--member serviceAccount:ass1-sa@${PROJECT_ID}.iam.gserviceaccount.com \
--role roles/owner
```

Note: If you receive an error message like '(gcloud.projects.add-iam-policy-binding) unrecognized arguments: ass1-sa@. . . .' the cause may be a space after `serviceAccount:` in the command above.

And create the service account key (again, don't replace anything):

```
gcloud iam service-accounts keys create ~/ass1-key.json \
--iam-account ass1-sa@${PROJECT_ID}.iam.gserviceaccount.com
```

This creates a private key stored in a JSON file named *ass1-key.json* in your home directory. Set an environment variable for your service account key for this session:

```
export GOOGLE_APPLICATION_CREDENTIALS=~/ass1-key.json
```

Remember to repeat this last step if your session becomes disconnected.

1.4 Test the app locally

Make sure you are in 'lab2-assignment/app'. We will create an isolated Python 3 environment with virtualenv:

```
virtualenv -p python3 env
```

Enter the environment you just created named *env*

```
source env/bin/activate
```

To deactivate your virtual environment, type `deactivate`. Now use *pip* to install the dependencies for the project, listed in `requirements.txt` to the virtual environment.

```
pip install -r requirements.txt
```

We won't fully launch the app, but we need to create an app engine instance to be able to use Datastore. Execute

```
gcloud app create
```

and select a region that supports flexible apps.

We will ceate a cloud storage bucket with the same name as the project. First, create an environment variable:

```
export CLOUD_STORAGE_BUCKET=${PROJECT_ID}
```

Then create the Cloud Storage bucket with *gsutil*, just like we did in Lab 1 (don't replace anything, we are again referring to the environment variable `PROJECT_ID`).

```
gsutil mb gs://${PROJECT_ID}
```

Finally, we can run the application.

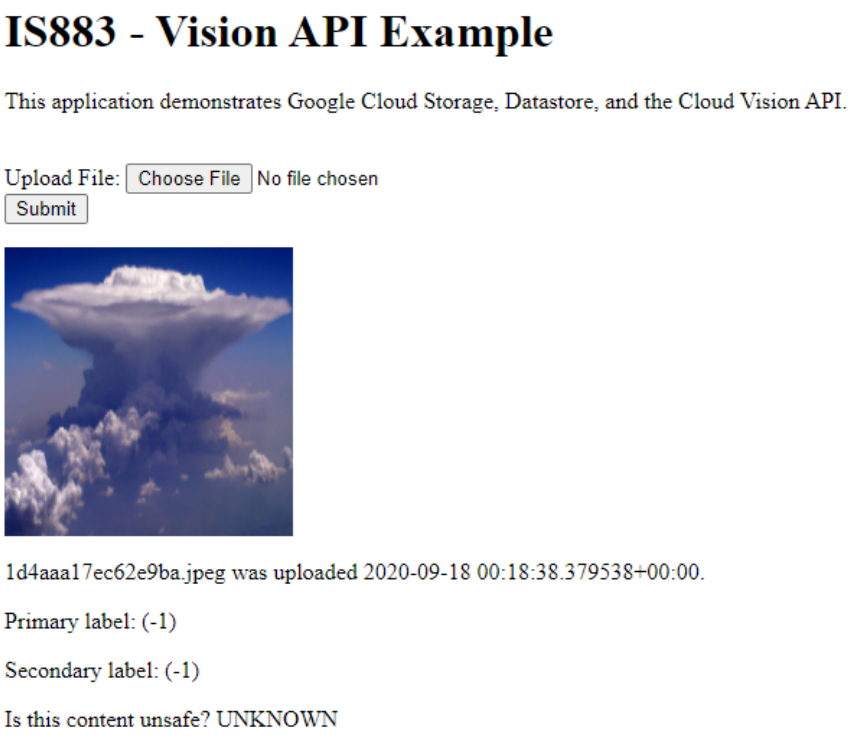

Figure 1: Before starting assignment.

```
python main_assignment1.py
```

If you completed the setup successfully, a web server will start with a link that you can visit (`127.0.0.1:8080`). You should see a simple web page with a place to upload images. Try uploading an image. The output will be similar to Figure 1: You'll notice the image is displayed after being uploaded, but it does not have labels associated with it nor a classification of whether or not it contains explicit content.

You can interupt the web server by typing 'CTRL + C' in the shell and hitting Enter.

1.5 Picking up where you left it

Picture this: You slog away for hours before finally getting the web app running. You go for dinner to celebrate. When you return, you find that things which were once working again seem broken.

Do not despair, you do not have to repeat all the previous steps. All we have to do is reset the environment variables, and enter our virtual environment. Executing the following lines should get you running again (only replace `[your-project-id]` with your project id - do not replace anything else):

```
export PROJECT_ID=$(gcloud config get-value core/project)
export GOOGLE_APPLICATION_CREDENTIALS=~/ass1-key.json
export CLOUD_STORAGE_BUCKET=${PROJECT_ID}
source env/bin/activate
```

The script called pickup.sh provided in the app folder already contain these commands. Navigate to 'IS883-21/lab2-assignment/app/', make sure that the environment 'env' is in this folder and execute

```
source pickup.sh
```

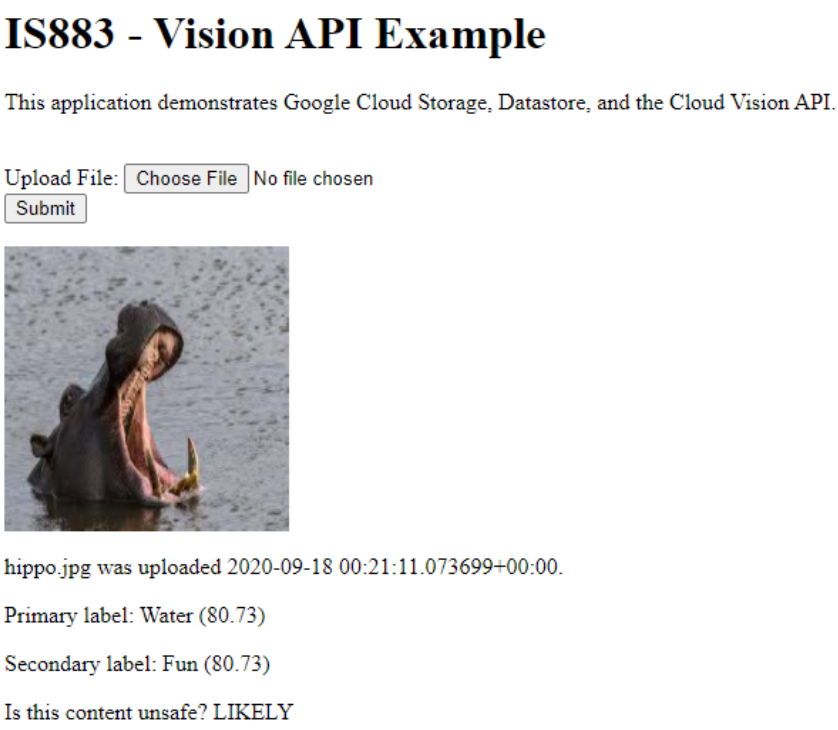

Figure 2: After completion.

2. The assignment

Your job is to edit the file `main_assignment1.py` so that the images are correctly stored and displayed. Figure 2 gives an example of the expected result. The file `main_assignment1.py` has three commented `TODO` statements - you should only enter new code within the denoted blocks, do not change anything else.

Remember to refer to the lab 2 python script and the documentation for the **Vision API** and **Datastore queries** for examples.

I recommend that you stop the web server, edit `main_assignment1.py` in your browser by clicking the 'Open Editor' button, then restart the web server `python main_assignment1.py`. This allows you to immediately test whether your changes had the desired effect.

2.1 Call the label detection API and store the two most likely labels

In the code block labelled `TODO 2.1`, call the vision labelling API to detect labels. Store the two labels with the highest confidences in the variables `first_label` and `second_label`, and their respective confidence levels in the variables `first_probability, second_probability`.

2.2 Call the SafeSearch API and compute an `unsafe` likelihood

In the code block labelled `TODO 2.2`, call the vision labelling API to determine whether the image contains explicit content. This returns (integer) likelihood scores in five categories, `adult, medical, spoof, violence, racy`, which we can interpret using the provided tuple `likelihood_name`.

We will define the likelihood that a variable is unsafe as its worst liklihood across the five categories, so an image that scores 'VERY LIKELY' in `violence` and 'VERY UNLIKELY' everywhere else will be denoted as 'VERY LIKELY' unsafe. Store the resulting likelihood in the variable `is_unsafe`.

2.3 Sort the results queried from the Datastore

You'll notice that by default the images are displayed alphabetically by filename. Edit `main_assignment1.py` by inserting code in the block labeled `TODO 2.3` so that the images are instead sorted so that the most recent uploaded image is displayed first.

Hint: Conceptually, you can think of this as similar to writing an SQL query to look up items from a database and sort them. Refer to the **Datastore query documentation** for examples.

3. Submit your solution

Submit your assignment by going to 'Assessments > Analytics Assignment' on the course page. You will see a fields where you should copy-paste the code you included within the code block for each of the parts of the assignment. There will also be a question that asks you to submit a screenshot of the app functioning.

Please do not delete you `main_assignment1.py` file, it may be required in the case that there are dificulties grading the assignments.

Cleaning up

Remember to stop/delete/undeploy resources you no longer need.

4. Hints and debugging

Most importantly, always read your error messages for hints about what went wrong.

Setting up the project the first time:

- Check that your project is associated with a billing accout (hopefully the one with the course credits)
- The necessary APIs are not enabled, you can check enabled services with `gcloud services list`
- The service account key is not created correctly: check if the service account json is in your home folder (`cd ~`). If it is there and you did not receive errors while creating it is is probably fine, go to the next bullet, otherwise repeat the steps in 1.3 after enabling the apis.
- The environment variables are not set correctly. Specifically, we need `GOOGLE_APPLICATION_CREDENTIALS`, `PROJECT_ID` and `CLOUD_STORAGE_BUCKET` to point to the right locations. You can get all the current environment variables with `printenv`, or search for a specific one with does not point to it `printenv | grep [name-of-variable]`, for example `printenv | grep PROJECT_ID`. IF the variable is not set, running this command will produce no output. Reset your variables as needed (refer to 1.5).
- Make sure you install the needed packages (the requirements.txt line) while in the virtual environment, and that you run the app from there. When you are in the virtual environment you will see `(env)` (or whatever you called the environment) at the start of your terminal prompt. You can enter the virtual environment with `source env/bin/activate` and exit with `deactivate`.

Once you've set the project up correctly once, the most common issues you'll run into are:

- When you restart a session you may be in the wrong project (you can check this by looking at the project id in your shell prompt) - refer to 1.1 to set the shell to the correct project.
- When you restart a session you will need to reset the environment variables - refer to 1.5.
- Be careful to make sure when you are in/outside the virtual environment. When you are in the virtual environment you will see `(env)` (or whatever you called the environment) at the start of your terminal prompt. The script in 1.5 enters the virtual environment, or you can enter with `source env/bin/activate` and exit with `deactivate`.

API calls

- The documentation linked above contains examples in various programming languages of how to call the APIs, you are welcome to refer to this. The lab2 scripts that we looked at in the class also give examples.
- The API response will likely be a JSON object with various fields/attributes, some pointing to lists. To see an example of what the response looks like you can print out the response you get in the terminal, read the documentation for examples, or use the API explorers provided in the documentation (for example the API explorer is at the bottom of this page, and an exampe response is given a little higher on the page).