Working_with_data

November 30, 2020

1 Working with Zooniverse data

For this workshop we will be working with the Zooniverse's data aggregation code `panoptes-aggregation`. This code is designed to work directly with the data dumps exported from projects `Data Export` page.

General documentation for the aggregation code can be found on https://aggregation-caesar.zooniverse.org/docs

1.1 Installing the code

1.1.1 Local computer

Install python 3, the easiest way to do this is to download the Anaconda build from: https://www.anaconda.com/download/. This will pre-install many of the packages needed for the aggregation code.

Open a terminal and run: `pip install panoptes-aggregation[gui]`

Download the folder containing the example data

1.1.2 Google colab

Google colab is a service that runs python code in the cloud

1. Sign into google drive
2. Make a copy of the example data in your own google drive (easiest way is to download and re-uplaod the folder)
3. Right click the `Working_with_data.ipynb` file > Open with > Google Colaboratory
4. if this is not an option click "Connect more apps", search for "Google Colaboratory", enable it, and refresh the page.
5. Run the cell below to install the needed packages (it will take a few mins to finish)

```
[ ]: !pip install panoptes_aggregation --quiet
```

5. Run the cell below to mount google drive

```
[ ]: from google.colab import drive
     drive.mount('/content/drive')
```

6. Run the cell below to change director to the example folder (adjust if you have renamed the example folder)

```
[ ]: import os
     os.chdir('/content/drive/MyDrive/agg_workshop_2020/')
```

1.2 Zooniverse data exports

Zooniverse projects provide a large amount of data to research teams. These data can be exported from the `Data Export` tab on a project's `Lab` page.

1.2.1 Classification export

This `csv` file has one row for every classification submitted for a project. This files has the following columns:

- `classification_id`: A unique ID number assigned to each classification
- `user_name`: The name of the user that submitted the classification. Non logged-in users are assigned a unique name based on (a hashed version of) their IP address.
- `user_id`: User ID number is provided for logged-in users
- `user_ip`: A hashed version of the user's IP address (original IP addresses are not provided for privacy reasons)
- `workflow_id`: The ID number for the workflow the classification was made on
- `workflow_name`: The name of the workflow
- `workflow_version`: The major and minor workflow version for the classification
- `created_at`: The `UTC` timestamp for the classification
- `gold_standard`: Identifies if the classification was made on a gold standard subject
- `expert`: Identifies if the classification was made in "expert" mode
- `metadata`: A `JSON` blob containing additional metadata about the classification (e.g. browser size, browser user agent, classification duration, etc…)
- `annotations`: A `JSON` blob with the annotations made for each task in the workflow. The exact shape of this blob is dependent on the shape of the workflow.
- `subject_data`: A `JSON` blob with the metadata associated with the subject that was classified. The exact shape of this blob is dependent on the metadata uploaded to each subject
- `subject_ids`: The ID number for the subject classified

1.2.2 Subject export

This `csv` file has one row for every subject uploaded to a project. This file has the following columns:

- `subject_id`: A unique ID number assigned to each subject as they are uploaded
- `project_id`: The ID number for the project

- `workflow_id`: The workflow ID the subject is associated with
- `subject_set_id`: The ID of the subject set the subject is connected to
- `metadata`: A `JSON` blob with the subject's metadata
- `locations`: A `JSON` blob with the URL to each `frame` of the subject
- `classifications_count`: How many volunteers have classified the subject
- `retired_at`: If the subject is retired this is the `UTC` timestamp for when it was retired
- `retirement_reason`: The reason why it was retired

1.2.3 Workflows export

This `csv` file has the information for every major version of a workflow. This file has the following columns:

- `workflow_id`: The ID number for the workflow
- `display_name`: The display name for the workflow
- `version`: The major version number (changes when a task is edited on the workflow)
- `active`: `true` if the workflow is active
- `classifications_count`: How many classifications have been made on the workflow
- `primary_language`: The language code for the workflow
- `first_task`: The task key for the first task
- `retired_set_member_subjects_count`: The number of retired subjects from the workflow
- `tasks`: A `JSON` blob showing the full workflow structure
- `retirement`: The retirement rules for the workflow
- `strings`: A `JSON` blob containing all the text associated with the workflow
- `minor_version`: The minor workflow version number (changes when text is edited on the workflow)

All other columns are not typically used and are for experimental or more advanced workflow setups.

1.3 How the aggregation code works

Aggregation is done in a three step process.

1. Configure `extractors` and `reducers`
    1. Check over configuration files and edit them as needed
2. `Extract` the data from each classification into a more useful data format (i.e. flatten and normalize the data)
3. `Reduce` the the extracts for each subject together (i.e. aggregate the data)

1.4 Processing the example Penguin Watch data

If you are comfortable using the command line, open a terminal and navigate the folder containing the example data. If you are working with the anaconda build on Windows you can using the anaconda launcher to boot up a `Jupyter lab` tab in your web browser. Use the left panel to navigate to the folder with the example data then launch a terminal.

A selection of each of the above export files for the Penguin Watch project has been provided for this workshop. You should make a new folder called `aggregation` to direct the output of these scripts. These steps outline how to use the command line to run all of the aggregation scripts, but a GUI is available by running `panoptes_aggregation_gui` from the terminal.

1.4.1 Run configuration script

The auto-config script will detect the shape of a project's workflow and select the default extractor and reducers to use. For this example we want to configuration files for workflow `6465` version `52.76`:

[1]:
```
!panoptes_aggregation config Penguin-Watch-Example-data-dumps/
↪penguin-watch-workflows.csv 6465 -v 52 -m 76 -d aggregation_results
```

```
Saving Extractor config to:
/Volumes/Work/agg_workshop_2020/aggregation_results/Extractor_config_workflow_64
65_V52.76.yaml

Saving Reducer config to:
/Volumes/Work/agg_workshop_2020/aggregation_results/Reducer_config_workflow_6465
_V52.76_point_extractor_by_frame.yaml

Saving Reducer config to:
/Volumes/Work/agg_workshop_2020/aggregation_results/Reducer_config_workflow_6465
_V52.76_question_extractor.yaml

Saving Reducer config to:
/Volumes/Work/agg_workshop_2020/aggregation_results/Reducer_config_workflow_6465
_V52.76_shortcut_extractor.yaml

Saving task key look up table to:
/Volumes/Work/agg_workshop_2020/aggregation_results/Task_labels_workflow_6465_V5
2.76.yaml
```

See `panoptes_aggregation config -h` for help text explaining each of these inputs.

This will crate four new files:

- `Extractor_config_workflow_6465_V52.yaml`: The configuration file for the extractors
- `Reducer_config_workflow_6465_V52.76_shortcut_extractor.yaml`: The configuration file for the shourtcut reducer
- `Reducer_config_workflow_6465_V52_question_extractor.yaml`: The configuration file for the question reducer
- `Reducer_config_workflow_6465_V52_point_extractor_by_frame.yaml`: The configuration file for the point reducer
- `Task_labels_workflow_6465_V52.76.yaml`: A file with a look up table that matches the workflow task keys with the text associated with them

1.4.2 Edit the extractor config file

Task `T4` was never used in the final project so it can be removed from the config file. Today we are not intrested in task `T0` tool `3` ("other") so we will remove it and it's subtask form the config file. The final file should look like:

```
extractor_config:
    point_extractor_by_frame:
    -   task: T0
        tools:
        - 0
        - 1
        - 2
    question_extractor:
    -   task: T1
    shortcut_extractor:
    -   task: T6
workflow_id: 6465
workflow_version: '52.76'
```

Converting this extractor config for use on Caesar All of the extractors available in within `panoptes_aggregation` can be used as "external extractors" on Caesar.

1. On the "Extractors" tab in Caesar click "Create Extractor" and select "External"
2. Enter a unique "key" to reference this extractor later
    - This value will be used in the reducer later and will show up in the data export
3. Enter the "URL" as `https://aggregation-caesar.zooniverse.org/extractors/<extractor name>?task=<task ID>`
    - The `<task ID>` value is found next to the `task:` key for each extractor in the config file
    - For our example this would be `https://aggregation-caesar.zooniverse.org/extractors/point_extractor_by_frame?task=T0`
4. Enter the "Minimum workflow version"
    - All version above this number will be passed through this extractor
    - `52.76` for the example above (the value next to `workflow_version:`)
5. Click "Create External extractor"

Note: For question tasks you will typically want to use the built in question extractor

1.4.3 Run the extractors

The extraction script will create one `csv` file for each type of extractor being used. In this case there will be two files created, one for `point_extractor_by_frame` and one for `question_extractor`.

See `panoptes_aggregation extract -h` for help text explaining each of these inputs.

[2]:
```
!panoptes_aggregation extract Penguin-Watch-Example-data-dumps/
↪penguin-watch-classifications-trim.csv aggregation_results/
↪Extractor_config_workflow_6465_V52.76.yaml -o example -d aggregation_results
```

```
Extracting: 100% |###############################################| Time:  0:00:00
```

1.4.4 Edit the reducer config files

There are no configuration parameters for the question reducer so that files does not need to be edited, but for the point reducer we should switch it from the default `DBSCAN` reducer to the `HDBSCAN` one. We are making this switch since the Penguin Watch subjects have a large depth-of-field that causes the point clusters to be different densities across the image.

We can also use this config file to set the `min_cluster_size` and `min_samples` keywords. Here are some good values to start with:

```
reducer_config:
    point_reducer_hdbscan:
        min_cluster_size: 4
        min_samples: 3
```

Converting the reducer config for use on Caesar All of the reducers available in within `panoptes_aggregation` can be used as "external extractors" on Caesar.

1. On the "Reducers" tab in Caesar click "Crate Reducer" and select "External"
2. Enter a unique "key" to reference this reducer later
   - This value is used by the rules later and will show up in the data export
3. Enter the "URL" as `https://aggregation-caesar.zooniverse.org/reducers/<reducer name>?<param 1>=<value 1>&<param 2>=<value 2>&<etc ...>`
   - For this example `https://aggregation-caesar.zooniverse.org/reducers/point_reducer_hdbscan?min_cluster_size=`
4. Expand the "Filters" section
5. Fill in the "Extractor keys" section as a list
   - This the key picked in step 2 of the extractor config
6. Pick how you want "Repeated Classifications" to be handled
   - Defaults to "keep first"
   - "keep all" can be useful at the testing/debugging stage
7. Click "Create External reducer"

Note: For question tasks you will typically want to use the built in stats reducer

1.4.5 Run the reducers

See `panoptes_aggregation reduce -h` for help text explaining each of these inputs.

Note: By default only if a volunteer classifies the same subject multiple times only the first one is used. This can be changed with the `-F` flag on the command line (e.g. `-F all` to keep all, `-F first` to keep first, `-F last` to keep last)

[3]:
```
!panoptes_aggregation reduce aggregation_results/question_extractor_example.csv␣
↪aggregation_results/Reducer_config_workflow_6465_V52.76_question_extractor.
↪yaml -o example -d aggregation_results
```

```
Reducing: 100% |################################################| Time:  0:00:00
```

[4]:
```
!panoptes_aggregation reduce aggregation_results/shortcut_extractor_example.csv␣
↪aggregation_results/Reducer_config_workflow_6465_V52.76_shortcut_extractor.
↪yaml -o example -d aggregation_results
```

```
Reducing: 100% |################################################| Time:  0:00:00
```

[5]:
```
!panoptes_aggregation reduce aggregation_results/
↪point_extractor_by_frame_example.csv aggregation_results/
↪Reducer_config_workflow_6465_V52.76_point_extractor_by_frame.yaml -o example␣
↪-d aggregation_results
```

```
Reducing: 100% |################################################| Time:  0:00:01
```

1.4.6 Plot the results

The final step is examining the results of the point clustering. A `jupyter notebook` named `plotting_functions.ipynb` is included with the shared files, this can be run by either opening the notebook directly or running the command `jupyter lab` in the folder containing the file, and opening it in your web browser. User `shift+enter` to run the code in each of the cells.

Note: you will have to adjust the file paths in the "Reading in the data" and "Plotting all the images" sections to match your file set up.

1.4.7 Understanding what the extraction and reduction files contain

The columns of the question extractor and question reducer files contain some data already described above and the counts for each of the possible answers for each question.

Point extractor There are `x` and `y` data for each of the four point drawing tools:

- `data.frame0.T0_tool*_x`: A list of the `x` data for each point created for `tool*`
- `data.frame0.T0_tool*_y`: A list of the `y` data for each point created for `tool*`

Point reducer In addition to the original point data:

- `data.frame0.T0_tool*_points_x` : A list of `x` positions for **all** points drawn with `tool*`
- `data.frame0.T0_tool*_points_y` : A list of `y` positions for **all** points drawn with `tool*`
- `data.frame0.T0_tool*_cluster_labels` : A list of cluster labels for **all** points drawn with `tool*`
- `data.frame0.T0_tool*_cluster_probabilities`: A list of cluster probabilities for **all** points drawn with `tool*`
- `data.frame0.T0_tool*_clusters_persistance`: A measure for how persistent each **cluster** is (1.0 = stable, 0.0 = unstable)
- `data.frame0.T0_tool*_clusters_count` : The number of points in each **cluster** found
- `data.frame0.T0_tool*_clusters_x` : The weighted `x` position for each **cluster** found

- `data.frame0.T0_tool*_clusters_y` : The weighted `y` position for each **cluster** found
- `data.frame0.T0_tool*_clusters_var_x` : The weighted `x` variance of points in each **cluster** found
- `data.frame0.T0_tool*_clusters_var_y` : The weighted `y` variance of points in each **cluster** found
- `data.frame0.T0_tool*_clusters_var_x_y` : The weighted `x-y` covariance of points in each **cluster** found

1.5 Other things to try

- Play around with changing the `min_cluster_size` and `min_samples` parameters to see how they change the detected clusters
- Read the various `csv` files into you favorite programming language and explore the data

[ ]: