ProtestDB

Data infrastructure

This resource defines setup of interfacing with the dataset in SQLite using the python library sqlalchemy.

See the documentation for the SQLite schema

SETUP

Setup a virtual environemnt:

```
pyvenv venv
```

Then, activate environment and install requirements:

```
source venv/bin/activate
pip install -r requirements.txt
```

Add a symbolic link to the `protestDB` module:

```
ln -s ‘pwd‘/protestDB venv/lib/python3.5/site-packages/protestDB
```

The above command assumes that the current directory is the project root folder.

Configurations are set in `alembic.ini`. Especially the following two fields are relevant to set:

```
# DB name:
db_name = protest_images.db

# image location:
image_dir = images
```

Where the first indicates the name of the database file located in the project root directory. The second variable `image_dir` indicates the folder in which the locally saved images are to be found.

Since images are referred in the database by their exact filename, all images can just be saved in a flat hierachy in the root of `image_dir`. This assumes that all filenames are globally unique.

Migrate schema

Schema needs to be migrated using `alembic` whenever schematic changes occur, such as a new table being added or columns being modified.

Migrations can be done from project root, using:

```
alembic revision --autogenerate
alembic upgrade head
```

Configure

Set the path for the `*.db` SQLite file in the `alembic.ini` file. The syntax is folowing the python configparser.

Usage

This module will create the database file given by `config.py` if it does not exists once the library is imported.

Example of extraction fully joined table of tags, images and label The tags have been expanded such that each possible tag value is a column and the entries are either True or False, essentially forming a sparse vector of tags for each row.

```
from protestDB import cursor, models

pc = cursor.ProtestCursor()

all_images = pc.getLabelledImages()

# `getLabelledImages accepts *args and **kwargs
# which are used by `filter` and `filter_by` respectively:

# images from UCLA:
ucla = pc.getLabelledImages(source="UCLA")

# images from either Bing or Google and
# label is `1`:
from sqlalchemy import and_, or_
bingog = pc.getLabelledImages(
    and_(
        or_(
            models.Images.source == "google",
            models.Images.source == "bing"
        ),
        models.Labels.label == 1
    )
)
```

Example insertion:

```
from protestDB.cursor import ProtestCursor
# create a cursor:
pc = ProtestCursor()

# insert image file using ‘insertImage‘ method:
pc.insertImage(
   path_and_name = 'example_image.png',
   source        = 'google search',
   origin        = 'test',
   url           = 'example.com',
   tags          = ['protest', 'africa', 'example', 'test'],
   label         = .5
)
```

The above, will also make insertions into `Tags` table and link them to the image through the `TaggedImages` table.

Example filtering:

```
# Get list of all images with the tag 'protest':
protestTag = pc.getTag("protest")

protest_images = protestTag.images
```

Below is an example of printing out the number of images for each tag in the database.

Example tag stats:

```
tags = pc.session.query(models.Tags).all()

for tag in tags:
    print("{:<15} {:d}".format(
        tag.tagName,
        len(tag.images))
    )
```

The above prints out a two column table where the first column has a fixed width of 15 and is left-aligned.

Example show image:

```
t = pc.getTag("fire")
img = t.images[0]
img.show()
```

See the class `ProtestCursor` in the file protestDB.engine for documentation on the possible parameters and their meaning.

Drivers

The `*driver.py` files in the project root generally has in common that they operate on the database through the `ProtestCursor` class, however, otherwise they generally don't have anything in common. They are intentionally silos representing different tasks on the database.

The following table provides an overview of the driver files:

file	Purpose
`annotator_driver.py`	Simple GUI for labelling images as protest or non-protest related
`amazon_input_driver.py`	Builds a CSV file compatible with MTurk
`amazon_input_sample_driver.py`	Generates combined sample of 2*N samples from two datasets
`luca_driver.py`	CLI for inserting Luca Rossi database from ECB protest into DB
`mturk_score_driver.py`	Computes the pairwise Bradley-Terry score on MTurk batch ouptut
`search_terms_driver.py`	Automate multiple search term configurations
`serp_driver.py`	CLI for scraping images
`ucla_comparisons_driver.py`	Used to insert scores from UCLA into the db based on csv file
`ucla_driver.py`	CLI for inserting UCLA dataset into DB
`ucla_score_driver.py`	Calculate images scores, plot and possibly write them to db

Serp Scraper

This code defines a commandline interface for scraping images.

Get usage information:

```
python serp_driver.py --help
```

Otherwise the general idea is to provide a path to a directory where the images scraped will be saved, the key words to be scraped and the search engines (currently supports google and bing). Look in the help for additional arguments

Limits

Bing has a limit of 210 images where google goes up to 800 in principle.

Usage

Minimum arguments

```
python serp_driver.py images --key_words "jenifer anistion" "cats"
```

All arguments

```
python serp_driver.py images --sr google bing --key_words "jenifer anistion" "cats" --n_imag
```

Luca Driver

This script is for insertion of image dataset from Luca Rossi into the schema format of the `protestDB`.

For general usage reference see:

```
./luca_driver.py --help
```

Arguments should include `--image-dir` and `--csv-file`, then the script will open a connection to the database via the `protestDB` wrapper and insert records and references to the images accordingly.

If `--destination-dir` is set, then the images will also be moved to this folder.

UCLA Driver

This script inserts from the UCLA dataset into the `protestDB`. For general usage see:

```
./ucla_driver.py --help
```

Serp Search terms scraper

This is a script built to automate multiple searches configured in a csv file in the following format:

search_term	search_engine	n_images	label
cats	google	300	1
dogs	bing	332	0

Just pass the path to the csv file as an argument to the script

Attention: **When used, it will automatically add it to the db!!**

Sample Chooser

This script is designed to select a sample to be annotated on mechanical turk. It works by, first pulling all the images that were annotated as being protest related (ProtestNonProtestVotes.is_protest == 1). Then it iterates through every image and computing the hamming distance to every other image available. If the distance is lower then the threshold set, it removes one of the images from the dataset. It then shuffle the result, prints the original hashes of those images (as in the db) and saves them locally in a folder that can be specifided.

The seed is also set to a default in order to be reproducible, but it can be changed

Usage

```
python sample_chooser images --dir_dest sample --seed 23023
```

Test Turk Input

This scripts intents to test certains properties desired on the mechanical turk input. Both the csv file and on the images that are sitting on amazon s3 bucket. The tests are as follow:

- no pair is made with the same image
- every image has exactly 10 pairs
- no image occurs more than 5 times in a single hit
- there are 1000 unique images
- all images come from Luca Rossi's database and were labeled as protest related
- all links on the s3 bucket are available

Annotator driver

This scripts is a very simplistic GUI interface for labeling images. The current commands are the following:

- Unrelated - Space
- Related - enter
- Going back - b

Usage

```
python annotator_driver.py images 0
```

images is the name of the folder where the images are contained and the second arguments specify is the script should save the results in the db (1) or not (0).

Amazon input driver

This CLI script will output a csv file compatible with the input provided to Amazon mechanical turk. The amazon input driver, `amazon_input_driver.py` requires either a path to a folder with filenames, or the names can be piped via standard input.

The two ways of invoking this script is illustrated in the following:

```
./amazon_input_driver.py --files <file with filenames>
```

or using standard input:

```
cat <file_with_filenames.txt> | ./amazon_input_driver.py
```

Annomaly detection

This script has two purposes. First is to calculate a divergency measure defined as "the percentage of votes that deviate from the most frequent vote across the whole data set". The second purpouse is, given a worker, visually inspects his votes.

Usage

To output in standard out using a csv like format the pairs "workerid" -> "divergency measure"

```
python annomaly_detection.py my_csv.csv
```

To visually inspect the votes of a given worker

```
python annomaly_detection.py my_csv.csv --worker_id 13412412
```

UCLA Comparison Driver

This guy is responsible for inserting the UCLA format like csv into the db as comparisons.

Usage

```
python ucla_comparison_driver.py my_csv.csv --db
```

UCLA Scores Driver

After having the UCLA comparisons in the db, you can use this script to calculate and save the scores in the db. The script will first plot the scores, and if set with the db flag, it will save the scores in the label table. Because it takes a hell of a time to compute the scores, this script will always output it first to a csv file. If you pass a csv file that already exists, will assume that those are the cached scores and will use those.

Usage

```
python ucla_scores_driver.py my_csv.csv --db
```

Communicating with the EC2 instances

SSH

```
ssh ubuntu@Public DNS
```

SCP

```
scp ubuntu@Public DNS:~/Thesis_2018/analysis/logs/UCLA_validation_log.csv UCLA_validation_lo
```