COMP 598 Homework 5 – Data Collection & Cleaning

30 pts
Assigned Oct 21, 2021
Due Oct 29, 2021 @ 11:59 PM

Please read the instructions carefully and follow the specifications of the assignment closely.

Cleaning data (15 pts)

In this assignment, you will be practicing your skills to clean up messy JSON entries. You will be cleaning files containing user posts, which are recorded as JSON data.

For this purpose, you have been provided with a sample input file **`example.json`**. Each line in the file is a JSON object (resembling a Python dictionary) which represents a single post. These JSON objects may or may not be valid, and may also contain field consistency issues that you will fix. Write a script named **`clean.py`** that accepts an input file and produces a cleaned output file. It should work when invoked with the following command:

```
python3 clean.py -i <input_file> -o <output_file>
```

The input file will contain one JSON dictionary per line, similar to the sample input. Please refer to **`example.json`** (located in the data/ folder of the HW template) to get a better idea of the input data. Remember: **example.json** is just an example, and your script will be tested on other input files as well. Your script will need to handle arbitrary JSON dictionaries - but you can assume they will have some or all of the following keys: `title` or `title_text`; `createdAt`; `text`; `author`; `total_count`; `tags`.

The output file should also have one JSON dictionary (i.e., post) per line.

Your script should do the following:

1. Remove all the posts that don't have either a `title` or `title_text` field.
2. For objects with a `title_text` field, rename the field in the output object to `title`.
3. Standardize all `createdAt` date times to the UTC timezone.
4. Remove all the posts with invalid date time that can't be parsed using the ISO datetime standard.
5. Remove all the posts that are invalid JSON dictionaries.
6. Remove all the posts where the `author` field is empty, null, or N/A.
7. The value in the `total_count` can only be type `int, float, str`. You must attempt to cast `float` and `str` to an `int` value. Some examples:
   - "3" → 3; "27" → 27
   - "twenty" → INVALID
   - 22.1 → 22; 22.9 → 22
   - If you are unable to cast `total_count` to int, remove the post.
8. Remove all posts if the type of `total_count` is anything else rather than `int, float, str.` If `total_count` is not present, you do not have to remove the JSON object; keep it in the output file.
9. The `tags` field should be a list of individual words (where each word does NOT contain a space). Any element that contains spaces should be split into separate words. If the tags field is not present, you do not have to drop the record; keep the record in the output file. In the input files we'll use, tags will always be a list. Do not worry about conversions.

   - E.g., if tags is ["golf", "tennis", "football games"], after processing it should be ["golf", "tennis", "football", "games"]

10. Posts that haven't been flagged for removal should be written to the output file in the order they appear in the input file.

Tips:

- Python provides a very powerful datetime package called… datetime
- When working with JSON data, the json package is your friend.

Unit Tests (15 pts)

You might have noticed this assignment does not come with unit tests - that is because this time, you are supposed to write your own tests to test the script you wrote previously.

You will write, at minimum, 6 unit tests for the script *clean.py* in part 1. Each unit test should test one of the following constraints:

1. Posts that don't have either "title" or "title_text" should be removed.
2. createdAt dates that don't pass the ISO datetime standard should be removed.
3. Any lines that contain invalid JSON dictionaries should be ignored.
4. Any lines for which "author" is null, N/A or empty.
5. total_count is a string containing a cast-able number, total_count is cast to an int properly.
6. The tags field gets split on spaces when given a tag containing THREE words (e.g., "nba basketball game").

Your unit tests should be written using the *unittest* package. All tests should be (test) methods contained in one CleanTest class, sitting in a file called *test_clean.py*. The template for this exercise already contains the file - it is under **`submission_template/test/test_clean.py`.**

For writing these tests, we'll rely on the idea of *fixtures* - properly mocked data targeting specific constraints. Putting it simply, fixtures are a way to test data within unit tests. When writing the tests, use the fixtures we provide (they are in the `fixtures` folder, inside the `tests` folder). Since there are 6 test requirements, there are 6 fixtures; each of them contains exactly one line that breaks the constraints we suggested, and hence removed/transformed by your code. For example, for the first test, the fixture in the JSON object has neither a "title" nor a "title_text" key (see the file named `test_1.json` in the template folder). Write a test that loads the fixture file and write an assertion of what is expected to happen given the broken input.

We provide you with 6 fixture files. You can write more if you want (no extra points for that; but this might be a good coding exercise).

Note that we don't specify HOW you test your code … only the contract that your test must follow.

Tips:

- We will test your code by substituting in our own data files (e.g., "test_2.json") and running your unit tests. So be sure to follow the guidelines above concerning

Submission Instructions

See the details in the hw5 directory in the homework code repository.