

Project 1 In-Class Phase One (Due April 4th)

The Redactor

Introduction

Whenever sensitive information is shared with the public, the data must go through a redaction process. That is, all sensitive names and other information must be hidden. We've seen police reports, court transcripts, and hospital records all containing sensitive information. Redacting this information is often expensive and time consuming.

Task Overview

In this project, you will use your knowledge of Python and Text Analytics to design a system to accept textual documents (either html, xml, or plain text) then detect and redact "sensitive" items in the documents. Below is an example execution of the program.

```
python3 redactor.py --input '*.html' \  
                    --input 'otherfiles/*.txt' \  
                    --names --dates --places --phones \  
                    --concept 'kids' \  
                    --output 'files/' \  
                    --stats stderr
```

Running the program with this command line argument should read all files ending with `.html` in the current folder and also all files ending in `.txt` from the folder called `otherfiles/`. All these files will be redacted by the program. The program will look to redact all names, dates, places, phone numbers, and street address. Notice the flag `--concept`, this flag asks the system to redact all portions of text that have anything to do with a particular concept. In this case, all paragraphs or sentences that contain information about "kids" should be redacted. All the redacted files should be transformed to pdf files and written to the location described by `--output` flag. The final parameter, `--stats`, describes the file or location to write the statistics of the redacted files. Below we discuss each of the parameter in more detail.

--input

This parameter takes a [glob](#) that represents the files that can be accepted. More than one input flag may be

used to specify groups of files. The file types that are supported by Redactor are html and text files. If a file cannot be read or redacted an appropriate error message should be displayed to the user.



--output

This flag should specify a directory to store all the redacted files. The redacted files, regardless of their input type should be writed to PDF files. You should create a describe your own numbering scheme.

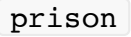
Redaction flags

The redaction flags list the entity types that should be extracted from all the input documents. The list of flags you are required to implements are:

- --names
- --genders
- --dates
- --places
- --addresses
- --phones

You are free to add you own! The definition of each of these types should be straight forward. In your README discussion file clearly give the paramaters you apply to each of the flags --- be clear what consitutes a place or an address. The redacted characters in the document should be replaced upon with a character of your chouce. Some popular characters include the unicode full block icon  or the ascii asterisk .

--concept

This flags, which can be repeated one or more times, takes one word or phrase that represents a concept. A concept could be either an idea or theme. Any section of the input files that refer to this concept should be redacted. For example, if the word was , a sentence (or paragraph) either containing the word or similar concepts, such as jail or incarcerated, that whole sentence or paragraph should be redacted. In your README file, make your definition of a concept clear. Also, clearly state how you create the context of a concept and justify your method.

---stats

Stats takes either the name of a file, or special files (stderr, stdout), and writes the a summary of the redaction process. Some statistics to include are the types and counts of redacted terms and the statistics of each redacted file. Be sure to describe the format of your outfile to in your README file.

Submission

For this project, supply one README document, code package, and compress it as a .tar.gz file. Ensure that your code can be, downloaded, extracted, and executed on the gpe1 machines.

Package your code using the setup.py and directory structure discussed earlier in the course. You should additionally add tests to this project submission.

```
from setuptools import setup, find_packages

setup(
    name='redactor',
    version='1.0',
    author='You Name',
    author_email='your ou email',
    packages=find_packages(exclude=('tests', 'docs')),
    setup_requires=['pytest-runner'],
    tests_require=['pytest']
)
```

In the tests folder, add a set of files that test the different features of your code. Tests do not have to be too creative but they should show that your code works as expected. There are several testing frameworks for python, for this project use the py.test framework. For questions use the message board and see the pytest documentation for more examples <http://doc.pytest.org/en/latest/assert.html> .

```
redactor/
  redactor/
    __init__.py
    redactor.py
  tests/
    test_download.py
    test_flagrecognition.py
    test_stats.py
    ...
  docs/
  README
  requirements.txt
  setup.py
```

Typing `python3 setup.py test` should execute your tests using the pytest-runner.

Remember not to include the contents of your env/venv files. It is not needed and can make grading and makes your `.tar.gz` file too large. Be sure to include your `requirements.txt` file. Upload a .tar.gz. to canvas.