Project 1 In-Class Phase Two (Due April 10th)

The Unredactor

Introduction

Whenever sensitive information is shared with the public, the data must go through a redaction process. That is, all sensitive names, places, and other sensitive information must be hidden. Documents such as 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 that accept plain text documents then detect and redact "sensitive" items in the documents. Below is an example execution of the program.

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

Running the program with this command line argument should read all files ending with `.txt` in 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 a 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 descuss 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. 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 writen to txt files. Each file should have the same name as the original file with the extension `.redacted` appended to the file name. If the output folder is in the same location as the input file.

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`
- `--addresses`
- `--phones`

You are free to add you own! Note: gender should be any term that reveals the gender of a person (e.g. him,

her, etc.). The other definitions of the rest of the terms 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 an address and phone number. The redacted characters in the document should be replaced upon with a character of your choice. Some popular characters include the unicode full block character `█` (U+2588).

--concept

This flag, which can be repeated one or more times, takes one word or phrase that represents a concept. A concept is 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 `prison`, a sentence (or paragraph) either containing the word or similar concepts, such as jail or incarcerated, that *whole sentence* 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. You may be creative here!

---stats

Stats takes either the name of a file, or special files (`stderr`, `stdout`), and writes 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.

Phase 2 Overview

As part of phase 2, you will be creating the *Unredactor*.
The unredactor will take a redacted document and the redaction flag as input, in return it will give the most likely candidates to fill in the redacted location.
The unredactor only needs to unredact names.
To predict the name you have to solve the Entity Resolution problem.

As you can guess, discovering names is not easy.
To discover the best names, we will have to train a model to help us predict missing words.
For this assignment, we will use the Large Movie Review Data Set. Please use this link to download the data set. This is a data set of movie reviews from IMDB containing. The initial goal of the data set is to discover the sentiment of each review. For this project, we will only use the reviews for their textual content. Below is some sample code to extract and print names from the movie reviews.

Python

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import glob
import io
import os
import pdb
import sys

import nltk
from nltk import sent_tokenize
from nltk import word_tokenize
from nltk import pos_tag
from nltk import ne_chunk


def get_entity(text):
    """Prints the entity inside of the text."""
    for sent in sent_tokenize(text):
        for chunk in ne_chunk(pos_tag(word_tokenize(sent))):
            if hasattr(chunk, 'label') and chunk.label() == 'PERSON':
                print(chunk.label(), ' '.join(c[0] for c in chunk.leaves()))


def doextraction(glob_text):
    """Get all the files from the given glob and pass them to the extractor."""
    for thefile in glob.glob(glob_text):
        with io.open(thefile, 'r', encoding='utf-8') as fyl:
            text = fyl.read()
            get_entity(text)


if __name__ == '__main__':
    # Usage: python3 entity-extractor.py 'train/pos/*.txt'
    doextraction(sys.argv[-1])
```

Note that in the code above we strict the type to PERSON. Given a new document such as the one below that contains at least one redated name. Create python code to help you predict the most likely unredacted name. If you are curious, the redacted name is Ashton Kutcher.

```python
'''This movie was sadly under-promoted but proved to be truly exceptional.
Entering the theatre I knew nothing about the film except that a friend wanted to see i

I was caught off guard with the high quality of the film.
I couldn't image ▇▇▇▇▇▇ ▇▇▇▇▇▇▇ in a serious role, but his performance truly
exemplified his character.
This movie is exceptional and deserves our monetary support, unlike so many other movie
It does not come lightly for me to recommend any movie,
but in this case I highly recommend that everyone see it.

This films is Truly Exceptional!'''
```

In order to discover the item redacted item, we need to create a function that takes a file with redacted items in it, and output the most likely name for each redaction.
Below is pseudocode the a naive solution:

```
# Training step:
for file in list_of_files:
    collect all entities in the file
    for each entity, increment the features associated with that entity

# Prediction phase
for file in list of files:
    extract all candidate entities
    extract features associated with each candidate entity
    Find the top-k matching entities
```

There are about 90K files available in the imdb data set split into test and train folders. Use the train folders to create features and evaluate your techniques. One of the most important aspects of this project is creating the appropriate set of features. Features may include, n-grams in the document, number of letters in the redacted word, the number of spaces in the redacted word, etc. Creative development of the feature vector is important for entity extraction.

The Google Graph Search Api may also prove useful. Using this, you can find a list of all important entities (Entity Search). Given a set of candidate matches you can use the information in google knoweldge graph to give you a ranked set of people.

For phase 2, we would like you to create a function to execute a redactor. Use the README to explain all assumptions. Be sure to give an examples of usage. Give clear directions. Add tests for each part of your code so we can better evaluate your code.

Note: there are several techniques for creating the unredactor. You can use a simple rule based approach, or create a classifier using the DictVectorizer in Sklearn. Use the message boards and office hours to discuss approaches. **We will primarily use the tests you create and your README file to evaluate your code.** Add tests that show the feature of your code that work and do not work. Unredacted texts should go in the output folder location.

Submission

For this project, supply one README document, code package, and compress it as a .tgz file. Ensure that your code can be, downloaded, extracted, and re-executed on your instance.

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

```python
from setuptools import setup, find_packages

setup(
    name='redactor',
    version='1.0',
    author='You Name',
    authour_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. Test 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 . This tutorial give the best discussion of how to write tests https://semaphoreci.com/community/tutorials/testing-python-applications-with-pytest .

```
redactor/
        redactor/
            __init__.py
            redactor.py
            unredactor.py
        tests/
            test_download.py
            test_flagrecognition.py
            test_stats.py
            test_train.py
            test_unredact.py
            ...
        docs/
        README
        requirements.txt
        setup.cfg
        setup.py
```

Note, the `setup.cfg` file should have at least the following text inside:

```
[aliases]
test=pytest

[tool:pytest]
norecursedirs = .*, CVS, _darcs, {arch}, *.egg, venv
```

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

If you use external APIs that require keys or special permission please include instructions on how we can also obtain access.

Please remove you env/venv files before submission. It is not needed and can make your `.tgz` file too large to grade. Be sure to include your `requirements.txt` and README files. Upload a .tgz to **Canvas**. This is for both inclass and the online section.

Extra links and Notes

Notes, to use some api's a larger instance may be needed. If you decide to use a larger instance please let us know and be sure to add this into your README.

Creating API Keys

https://cloud.google.com/docs/authentication/api-keys

Google NLP
https://googlecloudplatform.github.io/google-cloud-python/latest/language/usage.html#annotate-text