Readme--Elaine Dong edited by 3/28/2019

Projects Menu:

1. Summary\_checker\_nltk.py
2. Summary\_checker\_nlp.py
3. ExtractSocialActors.java
4. summary\_nltk\_checker.py

(the nltk version of checker)

Description:

This python program uses nltk as tool, and also uses social actor list to find the social actors (and also NNPs depending on your need) which is in the newspapers but is missing in the compilation of those newspapers.

The script loads the social actor list from the same folder (you can either run ExtractSocialActors.java to get it or download it from Github and uses it filter out unique social actor lists for both Compilation folder and Lynching folder. It filters out the social actors from both folders, and compares the filtered results. It adds any social actor that is present in the lynching newspaper, but is missing in their corresponding compilation. Finally, those missing was added and formed the output. Another output is the evaluation, it summarizes all the frequency of errored summaries, its percentage, the document that the type of error occurs, and also the summaries that contains the error.

Installation:

Make sure that you have installed python in your computer.

We need to install “nltk” to run the program.

Detailed steps:

1. Go to the terminal (for mac. If you use windows, go to the cmd.exe)
2. Enter “python –m pip install nltk” and follow the instructions that will show up in python. For instance, you may follow these instructions below:

“python” and enter

“import nltk” and enter

“nltk.download(‘punkt’)” and enter

“nltk.download (‘wordnet’)” and enter

“exit()” and enter

\*if “python” does not work, try “py” or “python3”. For instance: “py –m pip install nltk”

\* If it says “-bash: pip: command not found”, enter “sudo easy\_install pip”

\* If import nltk doesn’t work, try using the following commands:

1. python -m nltk.downloader averaged\_perceptron\_tagger

2. python -m nltk.downloader punkt

3. python -m nltk.downloader wordnet

How to run the program:

The program need to take four arguments:

1. the path of the lynching folder
2. the path of compilation folder
3. the output path
4. whether or not you need to detect proper nouns (1 for yes, 0 for no)

Example:

“python summary\_checker\_nltk.py ./ORIGINAL\_PDFTXTLynchingDBBYPCID/

./Compilations\ TXT/ ./ 1”

In this example, “.” Means the current directory. So the last argument “./” means output path is in the same directory.

As an example, it takes 30 seconds to run the whole program with “0” being the last argument.

Required Environment

Keep and social-actor-list.csv\*(see below) in the same folder.

Compilations TXT and Lynching articles do not have to be in the same folder, but it’s suggested to do it. Also, putting them under the same directory enables you to use “./ORIGINAL\_PDFTXTLynchingDBBYPCID/” and “./Compilations TXT/” as path.

\* social-actor-list.csv

The csv file is generated by a java program called “ExtractSocialActors.java”. (See explanations at the bottom). It contains a list of social actors. The csv file need to be downloaded with the programs.

Output:

“nltk-validity-checkNNP.csv” when 4th argument is 1;

“nltk-validity-noNNP.csv” when 4th argument is 0;

“summary\_checker\_nltk\_freq.csv”

Program technique:

It has a function that enables to check the missing proper nouns (Enter “1” in the fourth argument to enable this function). Proper nouns are detected as NNP and NNPS by using nltk’s pos tagger.

The program first loads the social-actor-list.csv and creates a list from it. Then it iterates through every folder in the article folder (your argument 1). For each folder, it also iterates through all documents inside, and filters out all the nouns (which are detected as NN, NNP, NNPS and NNS). Then it filters out all social actors from the nouns, and saves them in the form of lemmatized and lower cased words. If need to separate NNP (4th argument is “1”), it will filter out all NNPs and NNPSs from all non-social-actor words.

According to the folder name inside the article folder, we iterate through the compilation folder (your argument 2) to find the corresponding txt file. The program first tokenizes the content, and then filter out social actors in it. If need to separate NNP (4th argument is “1”), it will also filter out all NNPs and NNPSs from compilation.

After we get social actors list and possibly NNPs list from both article and compilation, we start to check missing ones for both cases. We compare the lemmatized lower case form of every word in the list, if the one from article does not appear in the compilation, we add it in the missing list. If the missing word has already been reported, we update the article source where we found this word. Finally we print out result.

1. summary\_corenlp\_checker.py

(StanfordCoreNLP version)

Description:

This python program uses stanfordcorenlp and stanfordner as tools, and also uses social actor list to find the social actors (and also NERs depending on your need) which is in the newspapers but is missing in the compilation of those newspapers.

The script loads the social actor list from the same folder (you can either run ExtractSocialActors.java to get it or download it from Github), and uses it filter out unique social actor lists for both Compilation folder and Lynching folder. It filters out the social actors from both folders, and compares the filtered results. It adds any social actor that is present in the lynching newspaper, but is missing in their corresponding compilation. Finally, those missing was added and formed the output. Another output is the evaluation, it summarizes all the frequency of errored summaries, its percentage, the document that the type of error occurs, and also the summaries that contains the error.

How to run the program

\*You also need to add “sudo” in front of the code.

1. the path of the lynching folder

1. the path of compilation folder
2. the output path
3. the path of StanfordCoreNLP folder (Download it from the official website <https://stanfordnlp.github.io/CoreNLP/download.html)>
4. whether or not you want to use NER to detect Named Entity Recognition (1 for yes, 0 for no)
5. path of the model (you can find it in the stanford ner folder. go to "classfier". you need the path of "english.muc.7class.distsim.crf.ser.gz" (REMEMBER IT IS CLASS 7! instead of 3 and 5. )
6. path of the jar (also in stanford ner folder. the path of stanford-ner.jar)

Here is my example from a Mac computer:

sudo  python3 summary\_corenlp\_checker.py /Users/ElaineDong/Box\ Sync/research/2018S-Dr.Franzosi/Python/COMPILATIONS\ Checker/ORIGINAL\_PDFTXTLynchingDBBYPCID /Users/ElaineDong/Box\ Sync/research/2018S-Dr.Franzosi/Python/COMPILATIONS\ Checker/Compilations\ TXT ./ /Users/ElaineDong/Box\ Sync/research/2018S-Dr.Franzosi/stanford-corenlp-full-2018-10-05 1 /Users/ElaineDong/Box\ Sync/research/2018S-Dr.Franzosi/stanford-ner-2018-10-16/classifiers/english.muc.7class.distsim.crf.ser.gz /Users/ElaineDong/Box\ Sync/research/2018S-Dr.Franzosi/stanford-ner-2018-10-16/stanford-ner.jar

Installation

Type “pip install stanfordcorenlp” in command line.

Follow the instructions to install stanfordcorenlp in python.

Download Stanford corenlp and ner

1. Stanford Corenlp can be found here:

<https://stanfordnlp.github.io/CoreNLP/>

You can directly drag to the terminal as argument 4 (for Mac).

1. Stanford ner can be found here:

<https://nlp.stanford.edu/software/CRF-NER.html>

You need to open the folder, the path to a file called “stanford-ner.jar” is argument 7.

You can also find a folder called “classifiers”. Find a GZ file called “english.muc.7class.distsim.crf.ser.gz”. The path to it is argument 6.

Required Environment

Keep and social-actor-list.csv\*(see below) in the same folder.

Compilations TXT and Lynching articles do not have to be in the same folder, but it’s suggested to do it. Also, putting them under the same directory enables you to use “./ORIGINAL\_PDFTXTLynchingDBBYPCID/” and “./Compilations\ TXT/” as path.

\* social-actor-list.csv

The csv file is generated by a java program called “ExtractSocialActors.java”. (See explanations at the bottom). It contains a list of social actors. The csv file need to be downloaded with the programs.

Output

“nltk-validity-checkNER.csv” when 5th argument is 1;

“nltk-validity-noNER.csv” when 5th argument is 0;

“summary\_checker\_corenlp\_NER\_freq”

Program technique:

The process is very similar as the summary\_nltk\_checker. When you do not need to check NER, the process is the same as not checking NNP in summary\_nltk\_checker.

When you need to check NER: Instead of filtering out NNPs, we now filter out location, date, organization, and person from the non-social-actor list from both article and compilation. Delete the NER from the article if its lower-cased lemma appeared in the compilation. Compare the two NER list, and add the missing ones to the missing list. Finally print out the list. Every time an error is encountered, its frequency is calculated. So the evaluation of frequencies is the other output.

1. ExtractSocialActors.java

(Program to create the social actor list)

Description:

Use Depth First Search (DFS) technique to search for social agents in Wordnet, by processing the hyponyms of “person” and “social group”.

This program creates a social actor list. Since we have already created one, you do not need to run it. You can just use the result of it. The output file called “social-actor-list.csv”, and you should keep it in the same directory as the checker program because it will read it as one of the inputs.

How to run the program

javac ExtractSocialActors.java

java ExtractSocialActors

Output

1. Social-actor-list.txt
2. Social-actor-list-verbose.txt

Required Environment

Keep WordNet-3.0 (it can be download here: <https://wordnet.princeton.edu/download> ) in the same folder.

Also keep a folder called “edu” in the same folder (you can download it from <https://projects.csail.mit.edu/jwi/> )

Program technique:

First get look up the word “person” and “social group” in the dictionary database, and get the word id of the two. Store the words to a stack so that we can do DFS in it. For each word that has been stored in this stack, we add their hyponyms (must be noun here) that have not been discovered yet to the stack as well as store them as social actor list. Then we delete this word that was used for searching. We keep repeating this process. Each time we use one word on the stack to find new hyponyms, and delete this word. We stop the program when the stack is empty, meaning that all the words stored in it have been used to find hyponyms, and we cannot find any new hyponyms. Finally, we can get all social actors. We print out only the words in social-actor-list.txt. We print out additional information such as the brief definition (“gloss”), and the count of it.