Usage of py_daisy and Open-Sesame

This documentation outlines the steps and the details of each step in annotating NewsScape data with PyDaisy and Open-Sesame. A short analysis and evaluation of the two libraries will be included at the end of each section.

Prior to Annotation

Copying the NewsScape data library

Before annotating the NewsScape data library with the two annotating libraries, I used the Python script `get_newsscape.py` to copy the files. These duplicate files are annotated to avoid irreversible changes to the original file during the development stage.

The following code shows the command that copies the 2019/01/01 NewsScape data files into the folder `newsscape/`.

```
python3 get_newsscape.py --year 2019 --month 01 --day 01 --folder_path
./newsscape
```

Preprocessing the data

The NewsScape data that is copied and will be processed are in `.seg` file format, which is the Red Hen Data Format. It is chosen because the file includes the metadata and the closed captions (which will be annotated).

Preprocessing is necessary for two reasons. First, we are only interested in annotating the closed captions in the `.seg` file so we have to extract it. Second, the closed captions are all in capital letters. Without processing, PyDaisy and OpenSesame cannot accurately annotate the sentence.

In other words, there are two preprocessing step:

1. Extract the closed captions for annotation
2. Preprocess the closed captions text so they can be recognized and annotated by PyDaisy and OpenSesame.

I created the script `convert_newscape.py` that extracts the closed captions, which is run by the following command:

```
python3 convert_newscape.py --path_to_file <filename>
```

NLP4J and BMST are used to perform lemmatization, part-of-speech tagging and dependency parsing as their combination in the pre-processing pipeline yields the best result of frame and frame element identification (Kabbach, Ribeyre & Herbelot, 2018).

```
pyfn/scripts/preprocess.sh -x 001 -t nlp4j -d bmst -p semafor
```

PyDaisy (Compatible with Berkeley FrameNet 1.7)

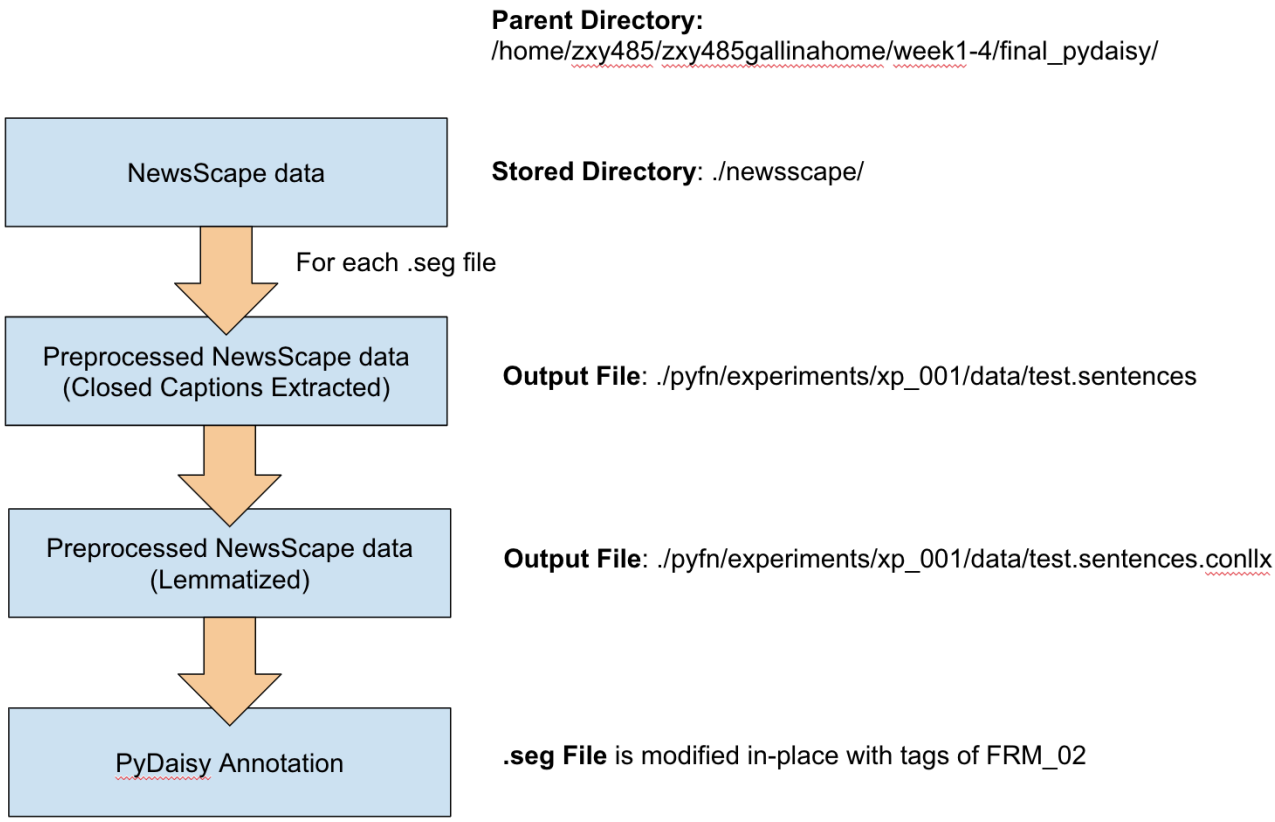
PyDaisy is the implementation of Disambiguation Algorithm for Inferring the Semantics of Y (Daisy) using Python for FrameNetBrasil. It can be accessed through the private GitHub repository - https://github.com/FrameNetBrasil/py_daisy (if granted access).

This section is structured into two subsections. The first section explains how to annotate the NewsScape dataset using the deployed Singularity container on Red Hen's HPC clusters. The second section elaborates on what I have worked on and the analysis of PyDaisy.

Instructions: Running the Singularity Container for PyDaisy FrameNet Annotation

The folder `/home/zxy485/zxy485gallinahome/week1-4/final-pydaisy` contains every script needed for annotation with PyDaisy. In particular, the singularity container `production.sif` contains the Python libraries such as NLTK necessary for running the annotating script `frm_02.py`.

The workflow is as shown by the following figure.



The respective commands for annotating a single NewsScape file is as followed:

```
# extracting the closed captions from the NewsScape datafile <filename>
singularity exec production.sif python3 /mnt/convert_newscape.py --path_to_file
/mnt/newsscape/mnt/rds/redhen/gallina/tv/2019/2019-01/2019-01-01/2019-01-
01_0000_US_CNN_CNN_Special_Report.seg

# preprocessing the closed captions with NLP4J and BMST
singularity exec production.sif /mnt/pyfn/scripts/preprocess.sh -x 001 -t nlp4j -
d bmst -p semafor

# Annotate with PyDaisy
singularity exec production.sif python3 /mnt/frm_02.py \
  --path_to_conllx /mnt/pyfn/experiments/xp_001/data \
  --path_to_file /mnt/newsscape/mnt/rds/redhen/gallina/tv/2019/2019-01/2019-01-
01/2019-01-01_0000_US_CNN_CNN_Special_Report.seg \
  --path_to_log /mnt/errors/log-2019-01-01_0000_US_CNN_CNN_Special_Report.seg
```

I wrap the commands in a Python script `annotate_dataset.py`, which can loop through and annotate every file in the `newsscape` folder. The commands are shortened to one line:

```
singularity exec production.sif python3 /mnt/annotate_dataset.py --path_to_folder
/mnt/newsscape
```

The sbatch script (`frame_annot.slurm`) used to submit the annotation job to the HPC clusters is as followed:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -c 1
#SBATCH --mem-per-cpu=5G
#SBATCH --time=3-00:00:00
#SBATCH --output=my.stdout
#SBATCH --error=my.err
#SBATCH --job-name="frame annotation"

module load gcc/6.3.0 openmpi/2.0.1 python/3.6.6
module load singularity
export SINGULARITY_BINDPATH="/home/zxy485/zxy485gallinahome/week1-4/final:/mnt"

singularity exec production.sif python3 -u /mnt/annotate_dataset.py --
path_to_folder /mnt/newsscape > ./output.out
```

My Modification of PyDaisy and Short Evaluation

The frame identification mechanism of PyDaisy is by first obtaining all the potential frames (including super-frames and sub-frames) represented by all frame-target words in the sentence, and subsequently finding the best cluster/combinations of frames. The ranking of clusters are based on the weighted relations between frames.

Instead of importing Berkeley FrameNet 1.7 that comes in XML format into a SQL database in order to use PyDaisy algorithms, I used NLTK Python library that comes with FN1.7 dataset to extract all the potential frames including the super-frames and sub-frames of the target word.

The major downside of adapting PyDaisy to Berkeley FrameNet 1.7 (FN1.7) is that the domain of FN1.7 is wider than FrameNet Brazil. This means that during the step of finding the best cluster of frames, there are exponentially more number of clusters due to the factorial growth rate of the combination function. To avoid the error of `MemoryError`, I have to perform two mitigations.

The first is through n-gram modeling of frame identification. In other words, the algorithm now store the rank values of the best cluster of frames for a window of fixed number of words. After iterating through the sentence, the algorithm picks the best frames for the set of respective frame-target words. Since frame identification involves inspecting the semantic meaning represented by the entire sentence, I speculate that this reductionist approach jeopardizes the accuracy of the frame identification.

The second mitigation step is by ignoring the stopwords in the sentence. That is, even though there are many frames available for stopwords such as `a`, `the`, etc., I avoid assigning frames to them because the sheer large number of frames associated with these frame-target stopwords increases the size of the clusters, which increases the running time of the algorithm.

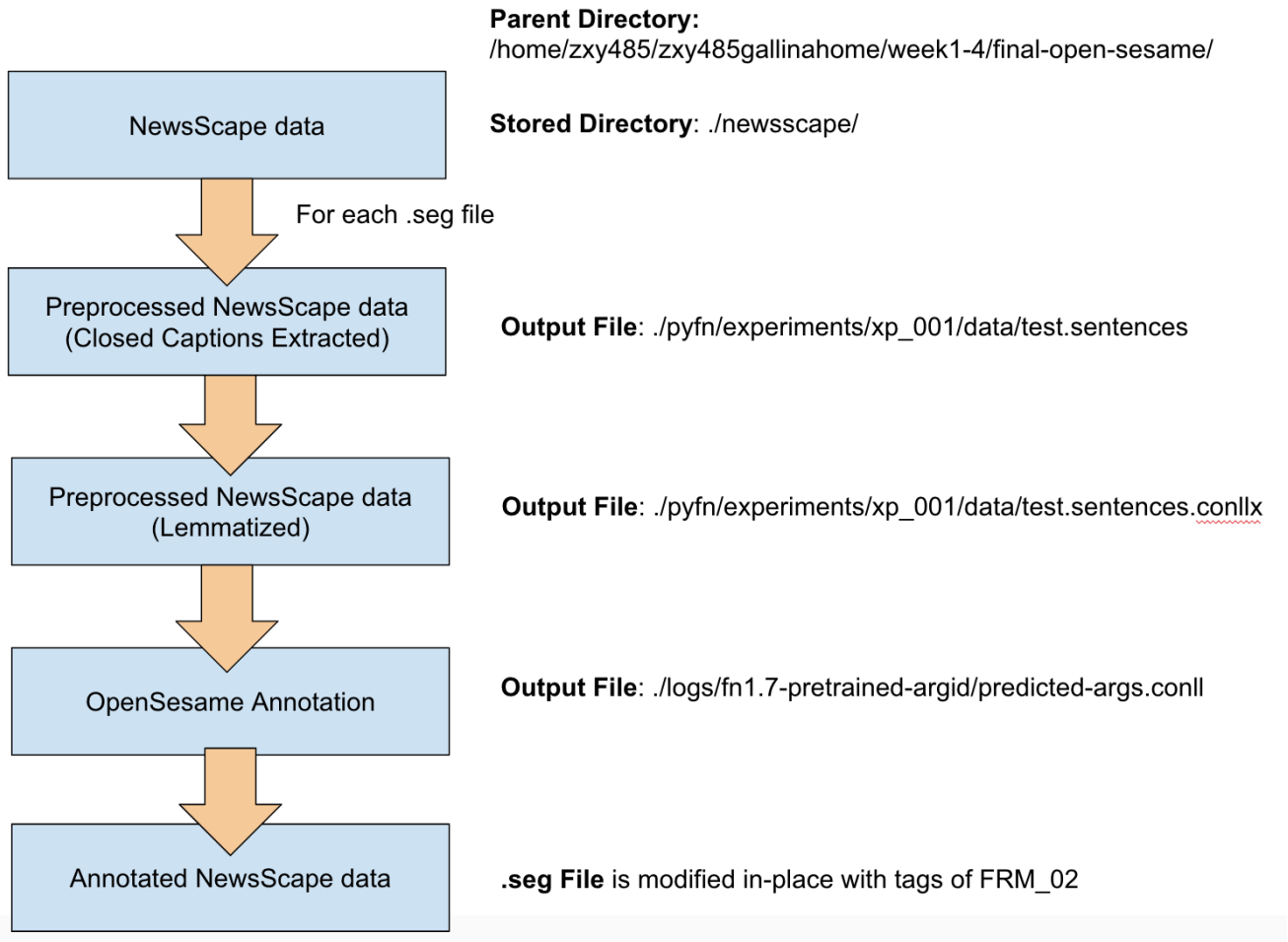
It should also be noted that FrameNet annotation involves frame and semantic role labeling, and PyDaisy only labels the frames. Because of the long running time due to the large number of clusters of frames and the potential inaccuracy that comes with the n-gram model, I turned to look at OpenSesame whose open-source codes include both frame and semantic role labeling.

Open-Sesame

Open-sesame is a parser that detects FrameNet frames and their frame-elements (arguments) from sentences using Segmental RNNs. This section is divided into two subsections. The first part elaborates on how to annotate the NewsScape dataset with OpenSesame, and the second part discusses the development of the annotating script `generate_RHL_format.py` and the challenges associated with the deployment.

Instructions: Running the Singularity Container for OpenSesame FrameNet Annotation

The folder `/home/zxy485/zxy485gallinahome/week1-4/final-open-sesame` contains every script needed for annotation with OpenSesame. In particular, the singularity container `production.sif` contains the Python libraries such as NLTK necessary for running the annotating script `generate_RHL_format.py`.



The respective commands for annotating a single NewsScape file is as followed:

```
# extracting the closed captions from the NewsScape datafile <filename>
singularity exec production.sif python3 /mnt/convert_newscape.py --path_to_file
/mnt/newsscape/mnt/rds/redhen/gallina/tv/2019/2019-01/2019-01-01/2019-01-
01_0000_US_CNN_CNN_Special_Report.seg

# preprocessing the closed captions with NLP4J and BMST
singularity exec production.sif /mnt/pyfn/scripts/preprocess.sh -x 001 -t nlp4j -
d bmst -p semafor

# Annotate with OpenSesame
singularity exec production.sif python3 /mnt/generate_RHL_format.py \
  --path_to_lemmatized_file
/mnt/pyfn/experiments/xp_001/data/test.sentences.conllx \
  --path_to_augmented_lemmatized_file
/mnt/pyfn/experiments/xp_001/data/test.sentences.processed.conllx \
  --path_to_open_sesame_input_file /mnt/sentences.txt \
  --path_to_open_sesame_output_file /mnt/logs/fn1.7-pretrained-argid/predicted-
args.conll \
  --path_to_seg_file /mnt/newsscape/mnt/rds/redhen/gallina/tv/2019/2019-01/2019-
01-01/2019-01-01_0000_US_CNN_CNN_Special_Report.seg
```

I wrap the commands in a Python script `annotate_dataset.py`, which can loop through and annotate every file in the `newsscape` folder. The commands are shortened to one line:

```
singularity exec production.sif python3 /mnt/annotate_dataset.py --path_to_folder
/mnt/newsscape
```

The sbatch script (`frame_annot.slurm`) used to submit the annotation job to the HPC clusters is as followed:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -c 1
#SBATCH --mem-per-cpu=5G
#SBATCH --time=3-00:00:00
#SBATCH --output=my.stdout
#SBATCH --error=my.err
#SBATCH --job-name="frame annotation"

module load gcc/6.3.0 openmpi/2.0.1 python/3.6.6
module load singularity
export SINGULARITY_BINDPATH="/home/zxy485/zxy485gallinahome/week1-4/final-open-
sesame:/mnt"
```

```
singularity exec production.sif python3 -u /mnt/annotate_dataset.py --
path_to_folder /mnt/newsscape > ./output.out
```

Development of `generate_RHL_format.py`

The `generate_RHL_format.py` is extended from the three commands that can generate predictions of frames and frame elements on unannotated sentences.

All the unannotated sentences are written in the text file `sentences.txt` separated by line break. They will passed through `sesame/targetid.py` , `sesame/frameid.py` and `sesame/argid.py` in order to generate a CONLL file that contains the predicted frames and frame elements.

Therefore, `generate_RHL_format.py` accomplished the three following tasks:

1. Copy the lemmatized sentences from `pyfn/experiments/xp_001/data/test.sentences.conllx` to `sentences.txt`
2. Run the three commands to generate the predictions of frames and frame elements.
3. Convert the CONLL file that contains the predicted frames and frame elements into Red Hen Data Format.

Note that `generate_RHL_format.py` does not process file by file; instead, it processes sentence by sentence. In other words, it only copies one sentence into `sentences.txt`, and after predicting the frames and frame elements, the script adds the frames and frame elements of the sentence into the `.seg` file.

Challenges with Deploying Open-Sesame on HPC Cluster

The main challenge comes with my unfamiliarity with modifying the environment and file relations when running the Singularity container. More specifically, the OpenSesame annotation step, which is based on the three commands that generate predictions of frames and frame elements, runs library module as script.

The three commands for generating predicted frames and arguments are as followed (copied from the OpenSESAME README.md).

```
# before modification
$ python -m sesame.targetid --mode predict \
                            --model_name fn1.7-pretrained-targetid \
                            --raw_input sentences.txt
$ python -m sesame.frameid --mode predict \
                           --model_name fn1.7-pretrained-frameid \
                           --raw_input logs/fn1.7-pretrained-targetid/predicted-
targets.conll
$ python -m sesame.argid --mode predict \
                         --model_name fn1.7-pretrained-argid \
                         --raw_input logs/fn1.7-pretrained-frameid/predicted-
frames.conll
```

However, when the scripts (`targetid.py` , `frameid.py` , `argid.py`) are mounted on the Singularity container, the exact commands would not work because of the path dependency. I had to change run the scripts just as scripts and not as packages.

```
# after modification
# scripts mounted on /mnt/sesame folder of Singularity container
$ python /mnt/sesame/targetid.py --mode predict \
                            --model_name fn1.7-pretrained-targetid \
                            --raw_input /mnt/sentences.txt
$ python /mnt/sesame/frameid.py --mode predict \
                           --model_name fn1.7-pretrained-frameid \
                           --raw_input /mnt/logs/logs/fn1.7-pretrained-
targetid/predicted-targets.conll
$ python /mnt/sesame/argid.py --mode predict \
                         --model_name fn1.7-pretrained-argid \
                         --raw_input /mnt/logs/logs/fn1.7-pretrained-
frameid/predicted-frames.conll
```

In addition, modifications are made to the following files to ensure that the path dependencies for reading and writing files remain valid after the files are mounted on the `/mnt/` folder of the Singularity container.

- `sesame/argid.py` :

  ```
  # line 34
  model_dir = "/mnt/logs/{}/".format(options.model_name)
  ```

- `sesame/frameid.py`

  ```
  # line 23
  model_dir = "/mnt/logs/{}/".format(options.model_name)
  ```

- `sesame/targetid.py`

```
# line 22
model_dir = "/mnt/logs/{}/".format(options.model_name)
```

- `sesame/globalconfig.py`

```
# line 5
config_json = open("/mnt/configurations/global_config.json", "r")
```

- `configurations/global_config.json`

```
{
    "version": 1.7,
    "data_directory": "/mnt/data/",
    "embeddings_file": "/mnt/data/glove.6B.100d.txt",
    "debug_mode": false
}
```

It is important to note this changes down because OpenSESAME may be updated and these changes have to be customly introduced after pulling the repository from GitHub.

Future Direction

After checking in with Prof. Tiago, who is my mentor, it seems that the argument identification of OpenSESAME is not precise (a lot of false positives). Therefore, in this week 6 (and possibly 7), I am looking into SEMAFOR and using its argument identification part to complement OpenSESAME's weakness.