Operating NLP Pipelines on the CLEW

DRAFT

Introduction

This is an investigation into three different workflows in LAPPS: Stanford, OpenNLP and GATE with a comparison of the performance of these three workflows. The workflow generally contains a tokenizer, sentence splitter, POS tagger, chunker and feature extractor. In this case, added tools have been implemented in LAPPS to correct the current problems for each built-in module. Also a script has been built to operate the whole process on a local server which then allows multiple inputs for training and testing purposes. Both pathways of using LAPPS or a local installation of the NLP components are installed on the CLEW. See the detailed instructions below.

1. LAPPS Workflow

To run the workflows in LAPPS, there is an installation of **Post Tokenizer**, **Post Sentence Splitter** and **POS Post Processor** in the CLEW. Generally, the workflows contain the following modules, consisting of the existing and the added HLA-G modules (see Figure 1).

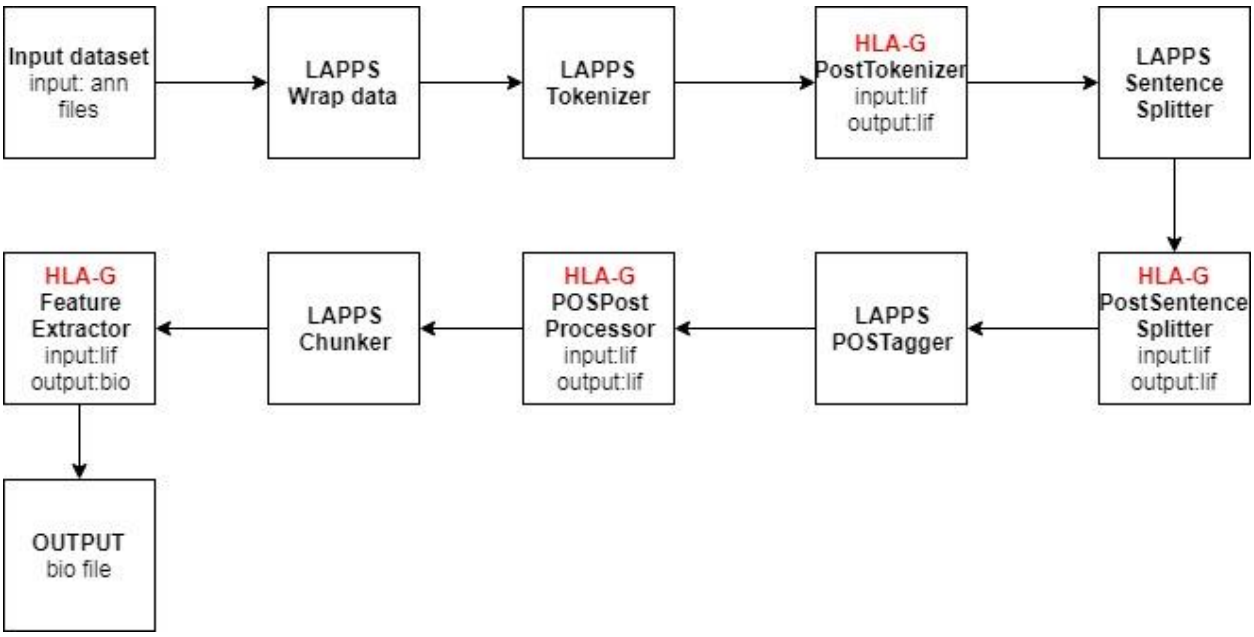

Figure 1. Processing modules of the generalized NLP pipeline.

The additional modules of the post-processors mentioned above have been installed in the CDC LAPPS machine. An example workflow has been created to test the correctness of all the modules. Workflows of the three pipelines are shared as public workflows in LAPPS which allows access to all users.

The three main workflows available at the moment are **Stanford workflow**, **OpenNLP workflow** and **GATE workflow**. The general process is similar except the built-in modules for each provider are used. The detailed description of each workflow including the modules we added is in the following subsections.

1.1 GATE Workflow

The complete workflow in GATE is shown in Figure 2.

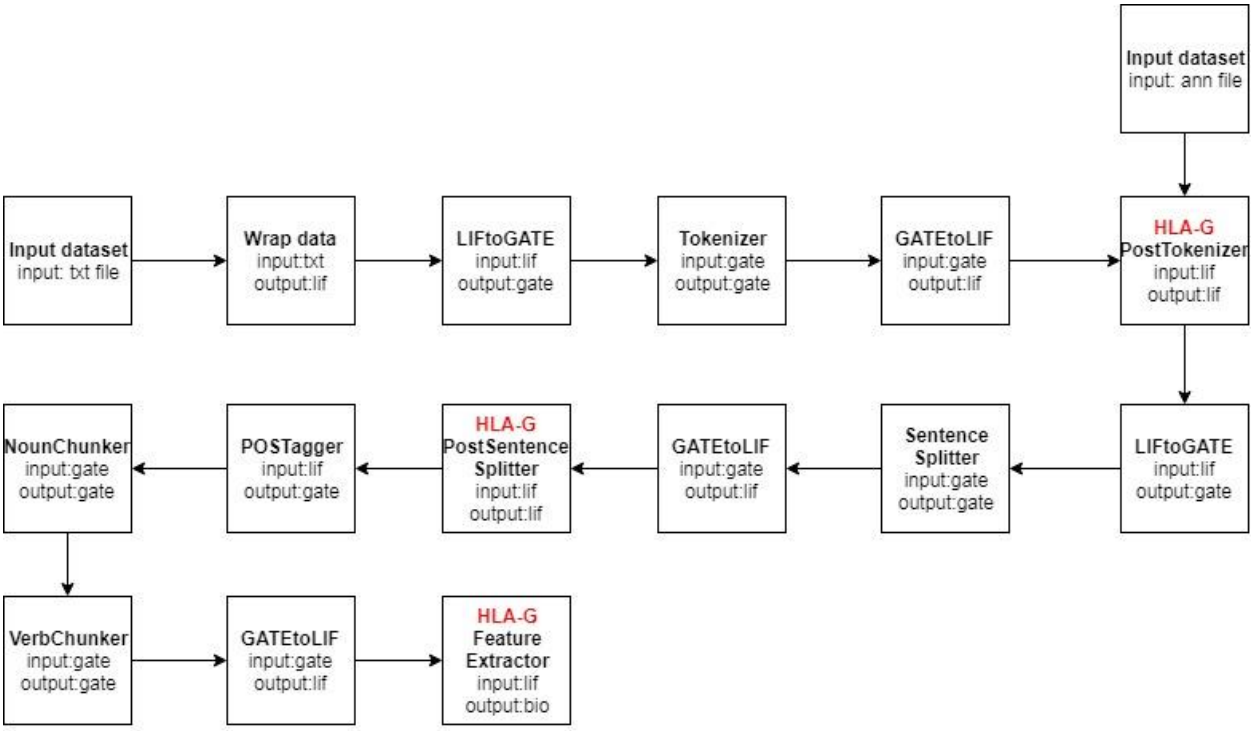

Figure 2. NLP pipeline for the GATE processing systems with modifications made by HLA-G.

The general process goes through tokenization, followed by sentence splitter, POS Tagger and Noun and Verb Chunker. The final step is a feature extractor we have implemented which extracts the results from the prior processing and outputs a BIO file. However, the performance on clinical texts in the GATE modules is defective in some ways so we have added several tools to make the results better fit clinical processing needs. A brief summary is provided below of functionalities implemented in these complimentary tools.

HLA-G_PostTokenizer-for-GATE

- Corrects the GATE Tokenizer problem of splitting a string of a digital number, like 0.1, into individual digits. The digits are kept as a single token.
- Keep alphanumeric serial numbers containing both numbers and capital letters as a single token.
- Read the annotation file to add a semantic tag feature into the LIF file indicating the semantic tag for each token.
- Save the result into a LIF file for further processing.

HLA-G_PostSentenceSplitter-for-GATE

- Separate the sentences by newline symbol as well as section heading detector and bullet point. Section heading is detected when the first letter for each word in the sentence (camel case).
- Save the result into a LIF file for future processing.

HLA-G_POSPostProcessor-for-GATE-Stanford-OpenNLP

- Get a list of medical nouns from an online site. Turn the POS tag of the tokens that are incorrectly tagged as NNP into NN.
- Save the result into a LIF file for future processing.

HLA-G_FeatureExtractor-for-GATE

- The user uses the feature selection interface to select all the features they want used to generate their language model. Subsequently those features typically require the following processing to be extracted:
    - Extract the token, length, orthography, token type and POS Tag information as features.
    - Extract the information of whether the token is inside a chunk and turn it into a feature.
    - Extract information of some tokens before and after the current token and add them into features for the current token.
    - Match the chunks and tokens with SNOMED CT concepts in Metathesaurus and get the SNOMED code. Add the SNOMED code to the feature set.
    - Extract the semantic tag result stored previously and add that as the tag in the final column of the output.
- Write the output into a BIO file.

1.2 Stanford and OpenNLP Workflow

- The workflow for Stanford and OpenNLP is similar to the GATE workflow including the corresponding PostTokenizer, PostSentenceSplitter, POSPostProcessor and FeatureExtractor. The processes of Stanford and OpenNLP workflow are illustrated in Figure 3.

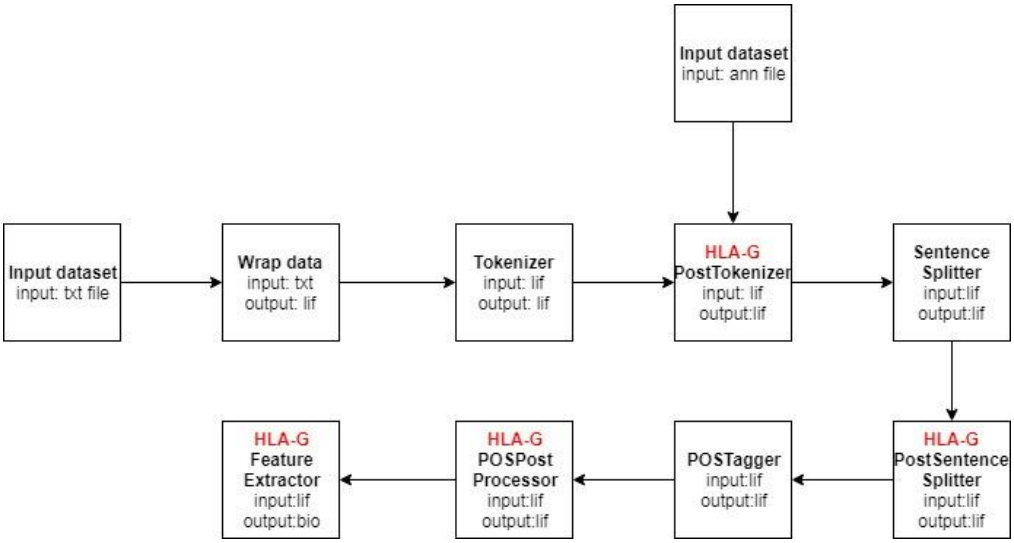

Figure 3. NLP pipeline for the Stanford and OpenNLP processing systems with modifications made by HLA-G.

- However, there are less features in these two workflows than the GATE workflow as some information including orthography is not available. The description of the FeatureExtractor is as follows:

HLA-G_PostTokenizer-for-Stanford – OpenNLP

- Split the tokens by hyphen, forward slash, quotes and other special symbols like '<', ':', etc.
- Save the result into the LIF file based on the structure of input LIF file for future processing.

HLA-G_PostSenteneSplitter-for-Stanford – OpenNLP

- Separate the sentences by newline symbols as well as section heading detector and bullet point. Section heading is detected when the first letter for each word in the sentence (camel case).
- Save the result to LIF file following the structure of input LIF file for future processing.

HLA-G_FeatureExtractor-for-Stanford-OpenNLP

This is the functions available in the feature extractor for the Stanford and OpenNLP pipelines.

- Extract the token, length, POS Tag information as features
- Extract information of some tokens before and after the current token and add them into features for current token.
- Match the tokens with SNOMED CT concepts in the Metathesaurus and get the SNOMED code. Add the SNOMED code to the feature set.
- Extract the semantic tag result stored previously and add that as the tag in the final column of output.
- Write the output into a BIO file.

1.3 Fixed Bug List

After investigating the whole workflow on some input files, the following bugs are found and fixed.

- In PostTokenizer for Stanford, the first couple of tokens representing *'{@value='* generated by the Stanford tokenizer are ignored and extents in the following tokens need to be changed otherwise the semantic tags would not be correctly matched.
- The order of the tokens in the output BIO file for Stanford and OpenNLP workflow is correct in python 3.6 but not correct in python 2.7.
- Extra newline tokens are added between section headings and sentences in GATE workflow.
- No chunker is available from the Stanford and OpenNLP teams in LAPPS so the GATE chunker was appended to these pipelines.

- In the output BIO file, the HL7 heading part has been excluded so that the training process will only focus on the main content of the report.
- The semantic tag name in the last column of BIO file has been shorten into one word followed by 'B-' or 'I-' since the CRF will recognize the features separated by space.
- The Feature Extractor has been changed to write the token features to BIO file sentence by sentence. Two sentences are separated by a blank line.

1.4 Issues Identified

After doing more investigation on different pathways above, some other issues were identified in the workflow that are not currently solved. The issues to be solved are summarized in the list below.

- **Parsers in LAPPS.** As described in the previous literature, the parser result could be included as features. An example feature could be the path linking the parent and children nodes for the current token. However, the built-in Stanford and OpenNLP Parser in LAPPS work based on the input text independent of the previous Tokenizer or Sentence Splitter result. It will perform its own Tokenizer, Sentence Splitter and POS Tagger which does not work well on our input texts. Therefore, to include parser results as features, a non-dependent parser needs to be built and installed which could take the results of the post sentence splitter and pass them to the feature extraction process.
- **Running time for training.** In order to evaluate the performance for three workflows, a batch of 500 files need to used for training and testing set. Currently the CRF training and testing is done outside of LAPPS. As well, using LAPPS to run many files is slow. So a separate implementation of the workflow is needed locally using the original source of the tokenizer, sentence splitter and POS Tagger with the HLA-G modifications for Stanford, OpenNLP and GATE respectively. Alternatively as a first step, a smaller number of files could be used to train the model and evaluate the performance of the three workflows first.

1.5 Future Investigations – Different pathways

Based on the current workflow, there are many other tools that could be investigated including other tools in LAPPS as yet unexplored. Different pathways could be investigated as follows.

1. Add chunker and MetaMap for Stanford and OpenNLP process.
2. Add lexical verification into the process. There is no built-in lexical verification found in the current POS Tagger or tokenizer systems. Medical lexical verification could be added.
3. Parser tools in LAPPS. There are Stanford, OpenNLP, DkPro Parsers available in LAPPS. Several information items related to the result of parser could be added as features.
4. Coreference tool in LAPPS. There are Stanford and DkPro Coreference tool available in LAPPS.
5. Relation Extraction in LAPPS. There is the ReVerb Relation Extractor available in LAPPS.
6. GATE Gazetteer in LAPPS.
7. Add tools available in other sources like nltk into LAPPS.

2. Operating a Pipeline on a Local server out of LAPPS

Due to the limitation of LAPPS, LAPPS does not scale well for a large number of input files. However, it calls different web service address for different existing modules This architecture was convenient for writing a single python script to call different services as well as our the introduced modules outside of LAPPS. Each web service called is executed and returns a LIF or GATE format string. This requires running the workflows on a local server to achieve scalability for a large number of input files. The local pipeline is installed on the CLEW. The processes of local pipelines are illustrated in Figure 4.

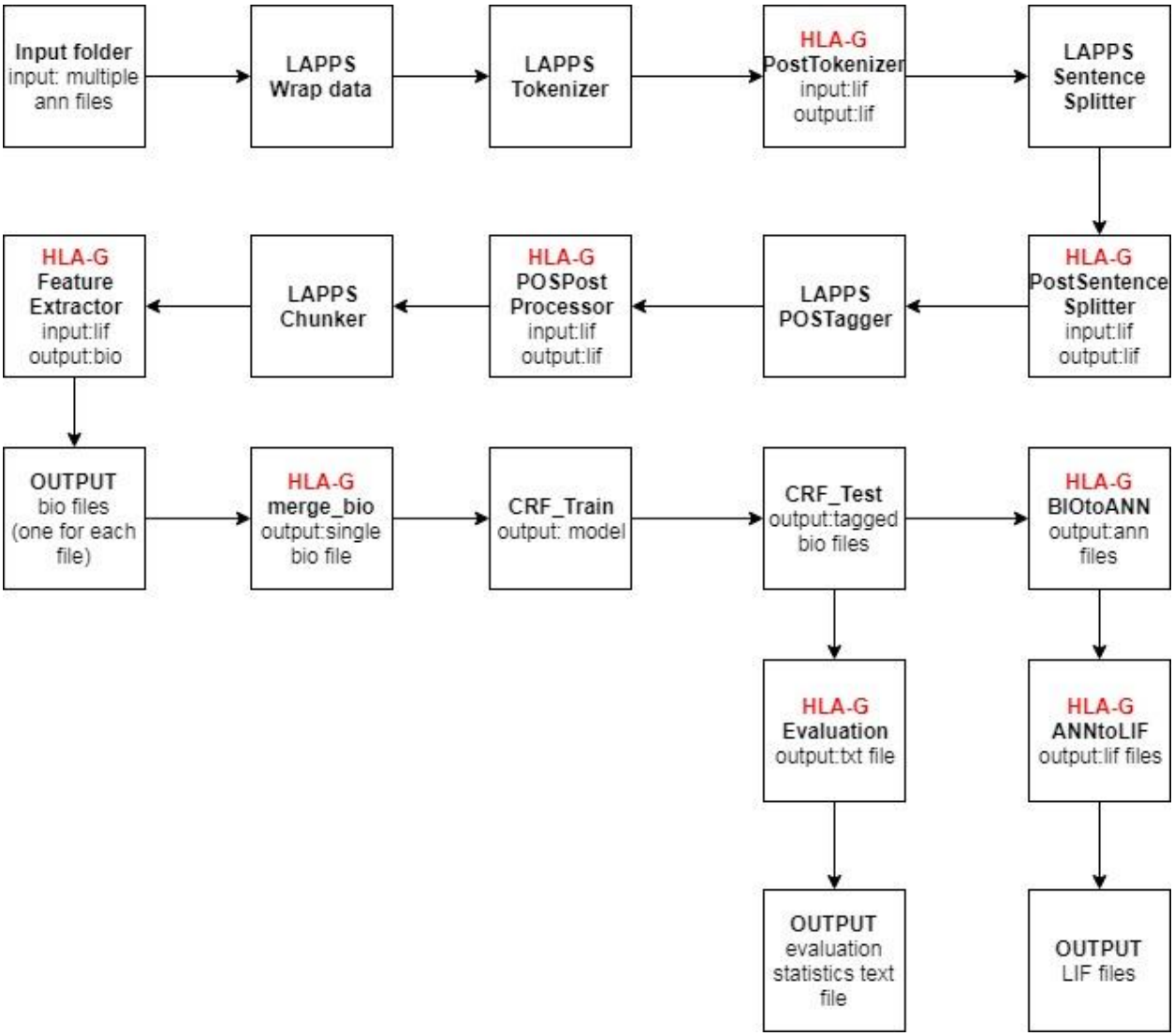

Figure 4. Processing architecture for complete end-to-end statistical machine learning NLP engine with an evaluation process.

The pipeline, outside of LAPPS, will take two input folders containing matched txt files and annotation files respectively. It will call different modules as well as pre- and post-processors and the feature extractor. It will generate a BIO file for each input file as output. After that, it will merge all these BIO files and generate a single BIO file as required to initiate training in the CRF machine learning process.

The trained Language Model is then used to tag the training files so that the differences between the Gold Standard and the computational tagging can be identified. After all the files are tagged by the CRF process, the evaluation statistics including Precision, Recall, F-score are calculated using differences between the gold standard annotations and the computationally tagged BIO files. The tagged files and LIF files are also generated from the tagged BIO files in this local pipeline.

3. Operational Use of the Pipelines

The script is placed in the **Pipeline** folder under the HLA-G folder in CLEW. To run the pipeline, the whole folder needs to be copied.

The pipeline script can be run using the following command:

```
cd Pipeline/Training

nohup python3.5 –u GATE.py > gate.out &

nohup python3.5 –u Stanford.py > stanford.out &

nohup python3.5 –u OpenNLP.py > opennlp.out &
```

At the moment, the pipeline takes the txt and ann files of batch 1 as input. The input files are placed in the following folder.

```
Pipeline/Training/input/CDC_batch_1_ann

Pipeline/Training/input/CDC_batch_1_txt
```

The outputs generated including merged bio file as well as bio files for each single input file is written into the following folder.

```
# Stanford Pipeline:
Pipeline/Training/output/bio/stanford/stanford_train.bio
Pipeline/Training/output/bio/stanford/train/*.bio

# GATE Pipeline:
Pipeline/Training/output/bio/gate/gate_train.bio
Pipeline/Training/output/bio/gate/train/*.bio

# OpenNLP Pipeline
Pipeline/Training/output/bio/opennlp/opennlp_train.bio
Pipeline/Training/output/bio/opennlp/train/*.bio
```

After the whole local pipelines are finished including the CRF training and testing as well as LIF conversion process, the generated LIF files as well as evaluation statistics text files are located in the following folders:

```
LIF files:
```

```
1. Stanford Pipeline:
Pipeline/Training/output/stanford_lif
2. OpenNLP Pipeline:
Pipeline/Training/output/opennlp_lif
3. GATE Pipeline:
Pipeline/Training/output/gate_lif

Evaluation statistics files:
1. Stanford Pipeline:
Pipeline/Training/output/stanford_evaluation.txt
2. OpenNLP Pipeline:
Pipeline/Training/output/opennlp_ evaluation.txt
3. GATE Pipeline:
Pipeline/Training/output/gate_evaluation.txt
```

4. Description of LIF Formatted Annotation Output File

1. LIF Output files

After a document has travelled through the whole pipeline, the output is written as a LIF format file containing all the **tagged annotations**, **agreement**, **annotator** and **the original text**. The LIF file also contains a list of annotations, where each annotation is a json block containing the following information:

- **Extent**: the start and end point of the annotated content in the original text.
- **Type**: the semantic tag of the annotation.
- **Agreement**: the agreement between the pipeline output and the gold standard.
- **Annotator**: the list of annotators who have annotated this tag. Either manual annotator (gold) or the language model or both.
- **Content**: the annotated content of the tag.
- **Metadata**: the name of the pipeline that produced this LIF output file.

2. Original text

The original text included in the LIF output file contains some HL7 segments followed by the main content of the report. Some HL7 segments include MSH, PID, ORC, OBR, etc. There is no processing in our system to remove those HL7 segments.

The HL7 segments are usually delimited by one or more newlines depending on the original message. The main content of the report is usually multiple newlines following the HL7 segments. Also, the extents in all annotations described above include all the HL7 segments and newlines properly.

The section heading of the main content of the report is usually at the start of the line. The content under this section heading is written one or two newlines followed by the section heading. The extents of the annotations also include these newlines. Moreover, in the GATE pipeline, the newline symbol between section headings and contents are included as one explicit token in the training and testing phase.