August 2019

Clinical Language Engineering Workbench (CLEW) Lessons Learned Document

FDA U.S. FOOD & DRUG ADMINISTRATION

CDC

Contents

CLEW Team Members

Team Member Organizations	Team Members
Center for Biologics Evaluation and Researcher Food and Drug Administration	Taxiarchis Botsis – Project Co-Lead Mark Walderhaug – Project Co-Lead Kory Kreimeyer Abhishek Pandey Matthew Foster Richard A. Forshee
Cancer Surveillance Branch Division of Cancer Prevention and Control Centers for Disease Control and Prevention	Sandy Jones – Project Co-Lead Joe Rogers Wendy Blumenthal Temitope Alimi
Northrop Grumman	Steve Campbell – Project Manager Fred Sieling – Project Manager Marcelo Caldas Sanjeev Baral
Health Language Analytics Global (sub-contract with Northrop Grumman)	Jon Patrick
Engility Corporation	Wei Chen Guangfan Zhang Wei Wang
Vassar College (sub-contract with Northrop Grumman)	Keith Suderman

This report compiles lessons that were realized and captured while developing a natural language process (NLP) workbench. The lessons have been separated into three categories that represent overall general observations; tools, systems development, and testing; and NLP and machine learning pipeline and model development.

General Observations

- A permanent, reliable, and cross-agency host server would allow for the Clinical Language Engineering Workbench (CLEW) to be fully implemented, reviewed, tested, and deployed across multiple domains. This project used CDC's Research and Development Lab servers that are not open to staff outside of CDC, including FDA and contract staff, which required special access to complete the project.

- Software development collaboration across U.S. government agencies and other organizations did not meet expectations of the project team including connectivity, available software development tools, server hosting, scalability, administrative support, and usage policy. CDC's Informatics Innovation Unit (IIU) environment encountered administrative delays, limited server hosting options and no plans for scalability. As a solution, we switched to Amazon Web Services (AWS); however, there continued to be challenges.

- Many open-source tools, frameworks, libraries, and applications exist. Integrating some of those tools is not straightforward. The LAPPS Interoperability Format (LIF) helps in this area, but it was still challenging to integrate UIMA-based tools such as cTAKES into pipelines.

- Using modular architectures, development teams with diverse backgrounds can cooperate to build a software ecosystem using RESTful APIs to coordinate modules across Windows and Ubuntu operating systems, across multiple NLP and tool code bases, and across CDC and FDA use cases.

- Two years was not enough time to complete this project. After the environmental scan, literature review, and exploration of existing open-source tools and platforms, the two agencies spent a significant amount of time conceptualizing the best approach to meet varying NLP needs of researchers from across domains.

- Partnering for software development by directly funding an existing lead open-source developer was valuable because it identified problem areas, exposed issues quickly, and allowed faster progress.

- The cancer pathology use case for NLP requires ongoing commitment to develop solutions that meet all government objectives including accuracy, utility across the cancer community, and open source in the full software stack.

- NLP has made strides in multiple areas:

    - Speech recognition.

  - Text-to-speech and speech-to-text conversions.
  - Language translations.
  - Sentiment analysis.
  - Summarization.
  - Most solutions focus on a generic corpora. When using medical materials, they do not understand the specifics within a clinical domain.

Tools, Systems Development, and Testing

- For future development projects, it is critical that developers build and test code frequently on an independent server configured for the target environment.
- A repository manager for developers would allow for storage of all dependencies and common code in a centralized location. This would eliminate duplication of effort and allow developers access to a library of code to perform tasks such as read/write UIMA files and talk to LAPPS servers.
- Development of the format conversion from UIMA (cTAKES) to the LAPPS Interoperability Format (LIF) was challenging due to size of the cTAKES type system. The syntactic conversion was simple, but the semantic conversion of mapping the 270+ cTAKES types onto the handful of LAPPS types took a lot of consideration and input from domain experts.

To incorporate cTAKES into the CLEW, a Java API had to be developed and made accessible as either a web service or part of a domain-specific Java application. To accomplish this, we attempted to incorporate cTAKES into a Maven project via a simple POM dependency. However, a bug in cTAKES versions 4.0.0 and 4.0.1 prohibited it from being included as a Maven POM dependency, causing a “URI is not hierarchical” Java exception when an application attempted to load cTAKES through the POM mechanism. To resolve this issue, we implemented a two-part solution that included making minor modifications to cTAKES source code and generating a new JAR file to replace the distributed version, and placing additional resource files into the “resources” directory of the target project structure. A revision to the cTAKES source code would allow third-party developers to leverage cTAKES for integration with their domain application.

- For a better user experience, additional programming in eMaRC Plus would allow for handling cases that received a non-response or error from the CLEW service.
- Tweak the Cancer Coding Service after the CLEW is implemented.
- Quantitative estimation of NLP accuracy metrics via methods such as 10 fold cross-validation requires a scalable computing platform like Jetstream, or at minimum, a cloud provider like AWS to enable timely feedback.

- Developer tips for creating an API definition and model:
  - Create endpoints for a resource (not a function or remote procedure call). We used /annotation as the endpoint.
  - It is better to use HTTP verbs, such as the same root endpoint (/annotation) for both the POST and GET methods. The POST method creates a new annotation, and the GET method retrieves a given annotation.
  - The POST method returns a status of 202 instead of 200, since the annotation is not created immediately, but notifies the user that the report was “accepted or received”. For future enhancements, a validation check can be implemented to notify the user with a 400 code that the document is not acceptable.
  - The GET method could return a 200 code when the annotation is ready and returned to the user, or a 404 code if the document is not ready. Future development could enable the server to keep better track of requests and return a 500 code if the annotation failed or a 400 code if the document was not useable.
  - The POST method could accept the document as a direct text/plain payload instead of having to deal with xxx-form-url-encode.
  - LAPPS is not flexible when it comes to creating production-grade pipelines for execution, or support for training models. To implement each pipeline as a web service, we had to recreate the entire pipeline as code. We had the option of calling each task within LAPPS, but there was no benefit in doing so except when integrating across different frameworks.

NLP and Machine Learning Pipeline and Model Development

- **Supervised learning.** This is difficult because it requires annotated corpora to train the model. Subject matter experts (SMEs) in the field can create the annotated corpora, which can be time-consuming and repetitive.

- **Development of different NLP pipelines.** We analyzed the behavior of different NLP pipelines. We implemented corresponding post processors to correct errors and designed feature sets for model training. In terms of future improvements in the development process, the code could be more generalized to support various versions of Python, including Python 2.7 and Python 3. In addition, the number of temporary output files could be decreased to save disk memory for larger reports.

- **Asynchronous API implementation.** We implemented NLP services for each pipeline using the full-batch model trained. The way of implementing asynchronous APIs was learned from this project, where the transaction ID is returned instantly after a POST and the result can be retrieved by doing a GET afterwards using the transaction ID. The HTTP status code returned by POST and GET in different cases could be considered more specifically. Endpoints could be created for resources such as BIO file containing features.

- **CER result evaluation and experimental design.** We produced a list of CER results for various types of experiments including training on each batch, full-batch training, and cross-batch testing. For future improvements, more experiments on feature engineering could be conducted and more depth analysis could be done.

- **Use cases.** The use cases were defined to serve CDC’s and FDA’s most immediate needs. This limited the arena of experimentation that was available given the time restrictions on completing the project. A wider range of use cases would better test the infrastructure and be better grounding for getting more services commissioned.

- **Pilot project corpus.** CDC’s pilot project corpus was deliberately selected to narrow the task to a scale that was achievable in the time allowed. Four tumor streams were selected with samples drawn from five data suppliers. A more narrow set of tumor streams with more documents would have enabled more reliable results.

- **Tag set.** Defining the tag set is an important aspect of any project. The tag set can be expanded significantly to serve broad-scale coding solutions.

- **An iterative modeling process for corpus annotation.** Two staff performed the corpus annotation over a five-month period using an iterative modeling process that developed the language model in concert with their annotations. This process proved effective in completing the annotation process more rapidly.

- **Performance standards.** The language models were trained on batches separated by data providers. They showed significantly different performance standards: models trained on one provider’s data could achieve as little as 15% accuracy on another provider’s data. This emphasized the importance of building models with careful distribution across all data sources.

- **Unreliable outputs for pathology reports.** Analysis of the output of components in standard NLP pipelines showed that they produced unreliable outputs for pathology reports, creating the need for post processors to correct their most common systematic mistakes.

- **Testing on a variety of pipelines.** Testing the various NLP pipelines showed that CTAKES—which is specifically designed to process clinical reports—failed to perform better than standard NLP pipelines. This indicates a need to experiment with a variety of open-source pipelines.