August 2019

Clinical Language Engineering Workbench (CLEW) User Guidance Document

FDA U.S. FOOD & DRUG ADMINISTRATION

CDC

Contents

The findings and conclusions in this report are those of the authors and do not necessarily represent the official position of the author's agencies (CDC, FDA). The authors have no conflicts of interest related to this work to disclose.

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

List of Abbreviations

API .....Application Programming Interface
ASPE .....Assistant Secretary for Planning and Evaluation
CDC .....Centers for Disease Control and Prevention
CER.....Clinical Entity Recognition
CLEW .....Clinical Language Engineering Workbench
CPU.....Central Processing Unit
CRF .....Conditional Random Fields
cTAKES.....clinical Text Analysis and Knowledge Extraction System
GB .....Gigabyte
RAM .....Random Access Memory
ETHER.....Event-based Text-mining of Health Electronic Records
eMaRC Plus .....Electronic Mapping, Reporting, and Coding Plus
FDA.....Food and Drug Administration
HLA .....Health Language Analytics Global
HTML .....Hypertext Markup Language
ICD-O-3 .....International Classification of Diseases for Oncology, 3rd Edition
IIS.....Internet Information Services
LAPPS Grid .....Language Applications Grid
LM.....Language Model
MS SQL .....Microsoft SQL
NLP .....Natural Language Processing
NLTK .....Natural Language Toolkit
NCBO .....National Center for Biomedical Ontology
PCOR .....Patient-Centered Outcomes Research
PCORnet .....National Patient-Centered Clinical Research Network
PCORTF .....Patient-Centered Outcomes Research Trust Fund
POS .....Part of Speech
RESTful .....Representational State Transfer

Introduction

This document explains how to install and use the Clinical Language Engineering Workbench (CLEW) and products developed in the Centers for Disease Control and Prevention (CDC) and the Food and Drug Administration (FDA) pilots. They were completed as part of the Assistant Secretary for Planning and Evaluation (ASPE) Patient-Centered Outcomes Research Trust Fund (PCORTF) Natural Language Processing (NLP) Workbench Web Services project.

The goals were to:

- Develop a generalized NLP web service to convert unstructured clinical information to structured and standardized coded data.
- Pilot NLP Workbench Web Services using cancer data and blood products and vaccine surveillance data.
- Update the NLP Workbench Web Services based on pilot results and provide technical documentation that describes the requirements for expansion of the NLP Workbench Web Services to meet additional domain needs.

The primary project objective was to provide a mechanism to “translate” free-text data into a structured form researchers, federal agencies, and public health agencies can use for surveillance and research purposes.

The CLEW is an open platform environment that uses different techniques to resolve specific use cases. It provides clinical NLP services and open-source NLP and machine learning tools to develop, experiment with, and refine clinical NLP solutions. The infrastructure is created for sharing new tools with the wider clinical NLP community, assembling NLP tools into a processing workflow, and generating training files for feeding machine learning algorithms to develop language models.

During this project, researchers found that good, focused corpora for the given problem are required to achieve accuracy when using NLP, especially statistical NLP and machine learning models.

System Architecture

This project is a workbench that allows users with different levels of knowledge about NLP practices can learn and discover solutions, develop their own solutions, or use shared services or pipelines. The CLEW uses microservices as a core concept to develop a service-oriented architecture that is modular and flexible to allow for future expansion and customization to meet different needs. Figure 1 below describes the CLEW microservices architecture.

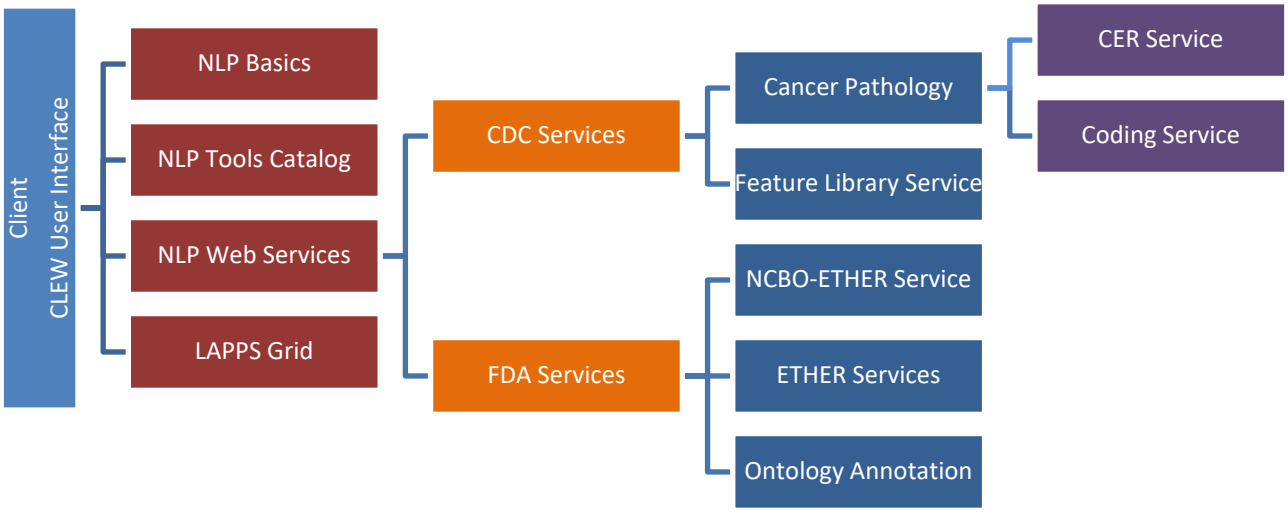

Figure 1. CLEW microservices architecture.

The CLEW's software components are basic and can run on multiple platforms. The third-party tool, The Language Applications (LAPPS) Grid, is better supported on Linux or Docker container environments. All other services run well on Windows or Linux. The CDC and FDA services run sufficiently on a machine with normal resources, for example two CPUs with 4 GB of RAM.

Description of Services

The CLEW User Interface

This is the front end of the CLEW. It hosts the information on NLP Basics and links to some of the other services on this list.

NLP Basics

This area provides basic NLP information for novice users. It provides detailed information on NLP rule-based and machine learning techniques, steps required to develop a statistical NLP model, and an example pipeline demonstration.

NLP Tools Catalog

This is a list of the open-source tools identified through the environmental scan and literature review.

NLP Web Services

In this area, test data can be sent to some services to see them in action. This service was conceived initially to wrap existing NLP framework solutions as RESTful web services. With the addition of LAPPS to the CLEW, this idea became obsolete, as LAPPS already performs this task. The project continues to be hosted to show past work.

LAPPS Grid

The CLEW instance of LAPPS Grid provides an environment for NLP experts to develop, test, evaluate, and share NLP and machine learning pipelines. LAPPS was developed as a collaborative effort among several universities (Vassar College, Brandeis University, Carnegie-Mellon University, and the Linguistic Data Consortium at the University of Pennsylvania) and is funded by the US National Science Foundation. It allows NLP pipelines to be created dynamically via drag and drop.

CDC Cancer Pathology Clinical Entity Recognizer (CER) Service

CDC's Cancer Pathology CER service implements a language model for pathology reports to identify histology, primary site, behavior, laterality, and grade.

CDC's Cancer Pathology Coding Service

Microservices map the terms extracted by the CDC Cancer Pathology CER service into ICD-O-3 histology and site codes.

Event-based Text-mining of Health Electronic Records (ETHER) Services

FDA services to extract clinical and temporal information using the ETHER application in combination with cTAKES and other NLP tools.

National Center for Biomedical Ontology (NCBO) BioPortal-ETHER Service

FDA service to extract clinical information using ETHER and NCBO BioPortal.

Ontology Annotation

Services that offer coding for FDA using third-party sites for ontology.

Processing Steps

Figure 2 below identifies the steps required to build an NLP machine learning model. The CLEW concentrates on a few tasks that enable users to accomplish the steps more efficiently.

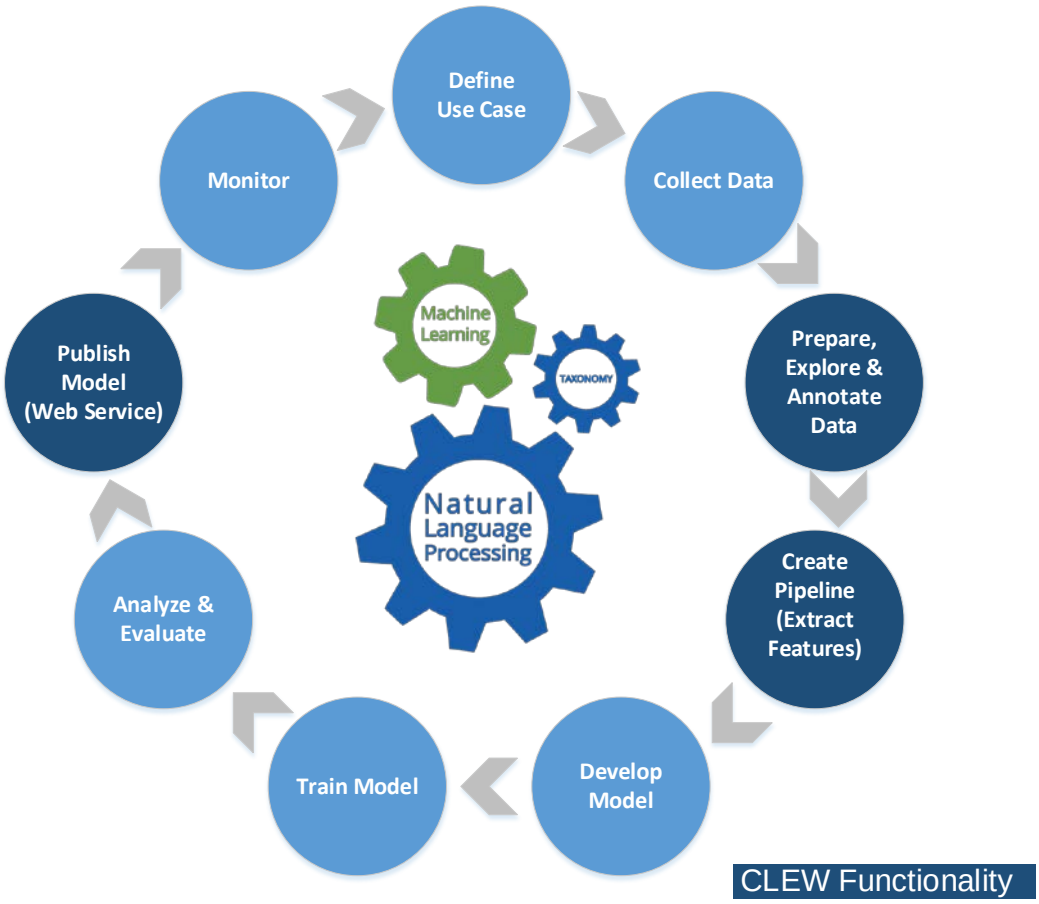

Figure 2. NLP machine learning processes.

NLP requires continuous evaluation of how the model is performing, retraining the model with new relevant data, and updating the data to keep the model current. Minor nuances in the data can drastically change the results or how the model functions. A standard NLP machine learning model development cycle is demonstrated in Figure 3 below.

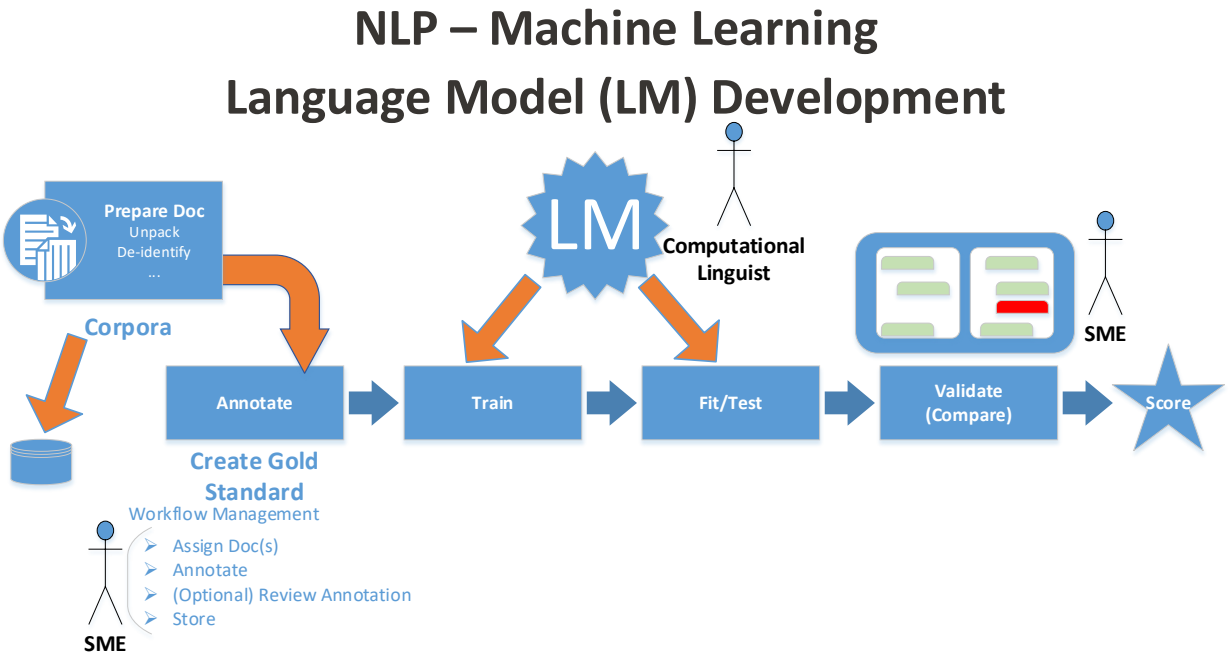

Figure 3. NLP – Machine Learning Language Model development cycle.

A gold standard model is important for an NLP solution to meet an acceptable level of accuracy.

Source Code and Instructions

All source code and instructions are provided at the following locations:

CLEW

CDC GitHub: https://github.com/CDCgov/NLPWorkbench

> Several projects included on the CDC GitHub are extra tools deployed within LAPPS Grid, including the HLALAPPSTools, cTAKES-clinical, cTAKES-relational, ETHER-clinical and ETHER-relational.

FDA GitHub: https://github.com/FDA/

LAPPS Grid

The LAPPS Grid project documentation is available at http://wiki.lappsgrid.org/Documentation.html and https://github.com/lapps.

> LAPPS Grid System Requirements: Linux – Ubuntu (preferred)

CDC Service Installation Instructions

This documentation includes installation instructions for all services developed as part of CDC's cancer pathology pilot. Five types of services are provided:

- HLA Post-Processors for LAPPS.
- CER pipelines.
- CER server.
- Cancer pathology coding service.
- Feature library web page.

HLA Post-Processors for LAPPS

Two Python packages are required:

- lxml
- requests

The source code is in CDC's GitHub under HLALAPPSTools at https://github.com/CDCgov/NLPWorkbench. Each module contains one Python file and one XML configuration file. Eight post-processors must be installed.

1. Copy the source code in HLAPostProcessors into the galaxy/mods/tools directory where the LAPPS tools files are stored.

```
# If you are in the home directory
cd galaxy/mods
cp –r /home/user/HLAPostProcessors tools
```

2. The tool configuration in the file galaxy/mods/config/tool_conf.xml must be extended with the HLAPostProcessors configuration. Add the new section configuration below to the configuration file:

```
<section id=”postprocessors” name=”HLAPostProcessors”>
 <tool file=”HLAPostProcessors/PostTokenizer_GATEDef.xml”/>
 <tool file=”HLAPostProcessors/PostTokenizer_StanfordDef.xml”/>
 <tool file=”HLAPostProcessors/PostTokenizer_OpenNLPDef.xml”/>
 <tool file=”HLAPostProcessors/PostSentenceSplitter_GATEDef.xml”/>
 <tool file=”HLAPostProcessors/PostSentenceSplitter_StanfordDef.xml”/>
 <tool file=”HLAPostProcessors/PostSentenceSplitter_OpenNLPDef.xml”/>
 <tool file=”HLAPostProcessors/POSPostProcessDef.xml”/>
 <tool file=”HLAPostProcessors/FeatureExtractor_Def.xml”/>
 <tool file=”HLAPostProcessors/FeatureExtractor_GATEDef.xml”/>
</section>
```

3. Restart LAPPS to enable the new tools. An example workflow demonstrating how these post-processors work has been shared in LAPPS. The command to run them locally is described in README.md.

CER Pipelines

This pilot is CDC's implementation of a language model to extract relevant information from cancer pathology reports. It was built using Python and Natural Language Toolkit (NLTK) and uses special algorithms for NLP machine learning training and testing.

Build requirements include:

- Python 2.7
- Apache Server
- Several Python dependencies

A machine with two CPUs and 4 GB of RAM can process about one document every 30 seconds, assuming the documents are one page in length. For training the model, a 64 Core machine must run for about 24 hours with a corpora of about 3000 annotated documents.

The following Python packages are required:

- lxml
- requests
- zeep
- subprocess
- corenlp
- nltk

All source code is provided at https://github.com/CDCgov/NLPWorkbench under HLACERPipeline.

The main installation process for CER pipelines is described below.

Install the Appropriate Python Packages

The CER pipelines run successfully on Python 3.5 +. Use the following pip3 command to install these packages for Python3.6:

```
sudo pip3 install lxml
sudo pip3 install requests
sudo pip3 install zeep
sudo pip3 install nltk
```

Install Conditional Random Fields (CRF)

The CRF source files are in the Training folder. To install them, run the command below inside the CRF++-0.58 directory. A C++ compiler (gcc 3.0 or higher) is required.

```
./configure
make
sudo make install
```

Install StanfordNLP Packages

Use the following command to download and extract the Stanford CoreNLP from the official releases:

```
wget http://nlp.stanford.edu/software/stanford-corenlp-full-2016-10-31.zip
unzip stanford-corenlp-full-2016-10-31.zip
```

You may need to install a file extraction utility.

```
sudo apt install unzip
```

Define the environment variable $CORENLP_HOME that points to the unzipped directory, and use the following command to install the Python Stanford CoreNLP package:

```
sudo pip3 install stanford-corenlp
```

After the installation is complete, use the following command to run the CER pipelines:

```
nohup python3 –u Stanford.py > stanford.out &
nohup python3 –u OpenNLP.py > opennlp.out &
nohup python3 –u GATE.py > gate.out &
nohup python3 –u cTAKE.py > ctake.out &
```

A more detailed description of the input and output format is in the README.md of the GitHub repository.

CER Server

The CER server takes the text of the report as input, uses the generated model to tag the file, and generates a file in LAPPS Interchange Format (LIF) with all of the tagged instances.

The following Python packages are required:

- lxml
- requests

- zeep
- bottle
- threading

All source code is provided at https://github.com/CDCgov/NLPWorkbench under HLACERServer.

1. Install the CER Pipeline.
2. To allow external access to the server, the CER server must be installed using Apache2, which is based on Python 2.7. Use the command below to install the packages required for the CER pipeline again with pip from Python 2.7:

```
sudo pip install lxml
sudo pip install requests
sudo pip install zeep
sudo pip install nltk
sudo pip install stanford-corenlp
sudo pip install bottle
```

3. Install Apache2 and setup the Stanford server, OpenNLP server and GATE server.

4. If the /etc/apache2/conf-available/httpd.conf configuration file does not exist, create it and add the following line to the Apache2 configuration /etc/apache2/apache2.conf.

```
IncludeOptional conf-enabled/*.conf
```

5. Add the following lines to the /etc/apache2/conf-available/httpd.conf file. Change the path to the wsgi file as needed.

```
# Server Address: http://hostname.com/cdc_service_stanford
WSGIDaemonProcess cdc_service home=/home/jenny/CDC_Server
WSGIScriptAlias /cdc_service_stanford /home/jenny/CDC_Server /stanford_server.wsgi
<Directory /home/jenny/CDC_Server>
   WSGIProcessGroup cdc_service
   WSGIApplicationGroup %{GLOBAL}
   Require all granted
</Directory>

# Server Address: http://hostname.com/cdc_service_opennlp
WSGIDaemonProcess cdc_service_opennlp home=/home/jenny/CDC_Server
WSGIScriptAlias /cdc_service_opennlp /home/jenny/CDC_Server /opennlp_server.wsgi
<Directory /home/jenny/CDC_Server>
   WSGIProcessGroup cdc_service_opennlp
   WSGIApplicationGroup %{GLOBAL}
```

```
   Require all granted
</Directory>

# Server Address: http://hostname.com/cdc_service_gate
WSGIDaemonProcess cdc_service_gate home=/home/jenny/CDC_Server
WSGIScriptAlias /cdc_service_gate /home/jenny/CDC_Server /gate_server.wsgi
<Directory /home/jenny/CDC_Server>
   WSGIProcessGroup cdc_service_gate
   WSGIApplicationGroup %{GLOBAL}
   Require all granted
</Directory>
```

6. Use the following command to restart Apache2:

```
sudo service apache2 restart
```

After the installation is finished, the server can be called using the script provided in the CDC_Server_Client directory.

Cancer Pathology Coding Service

This service matches text extracted from a pathology report using the HLACER Pipeline service above and produces ICD-O-3 coded output for primary site, laterality, histology, behavior, and grade.

Build requirements include:

- .NET 4.x
- IIS
- Windows 2016 Server
- MS SQL Server 2016 with SQL Management Studio:
  - MS SQL Server Express 2017 download location: www.microsoft.com/en-us/download/details.aspx?id=55994
  - MS SQL Server Management Studio 2017 download location: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017

Configure Windows 2016 Server to Serve ASP.NET based Web API

Add the following roles and features

1. Add the Web Server role; then under Web Server expand Application Development and select .NET Extensibility 4.6 and ASP.NET 4.6. See Figure 4 below.

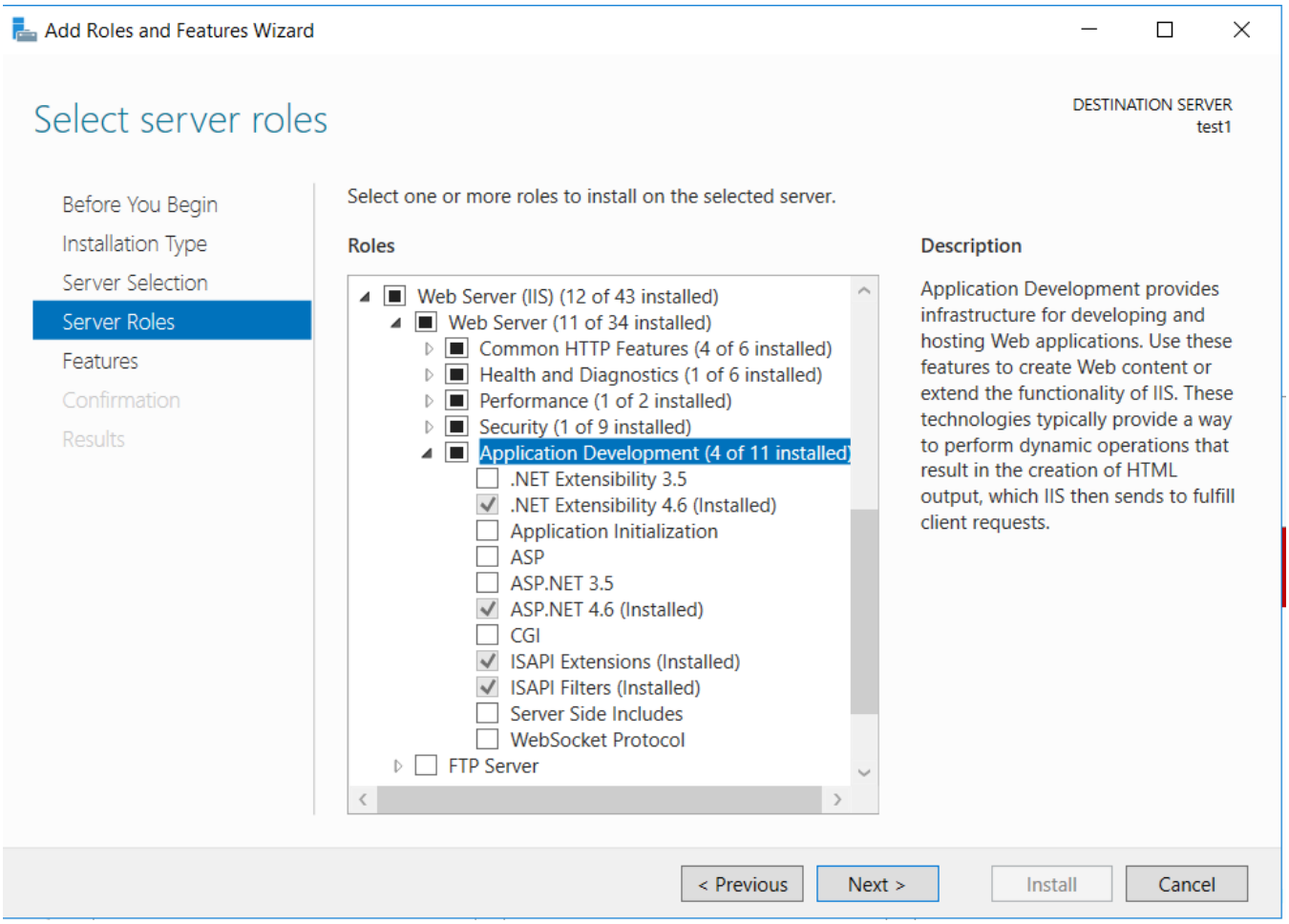

Figure 4. Screen shot of Web Server Role setup screen.

2. In Features selection step under .NET Framework 4.6 Features, check to see that ASP.NET 4.6 is selected. See Figure 5 below.

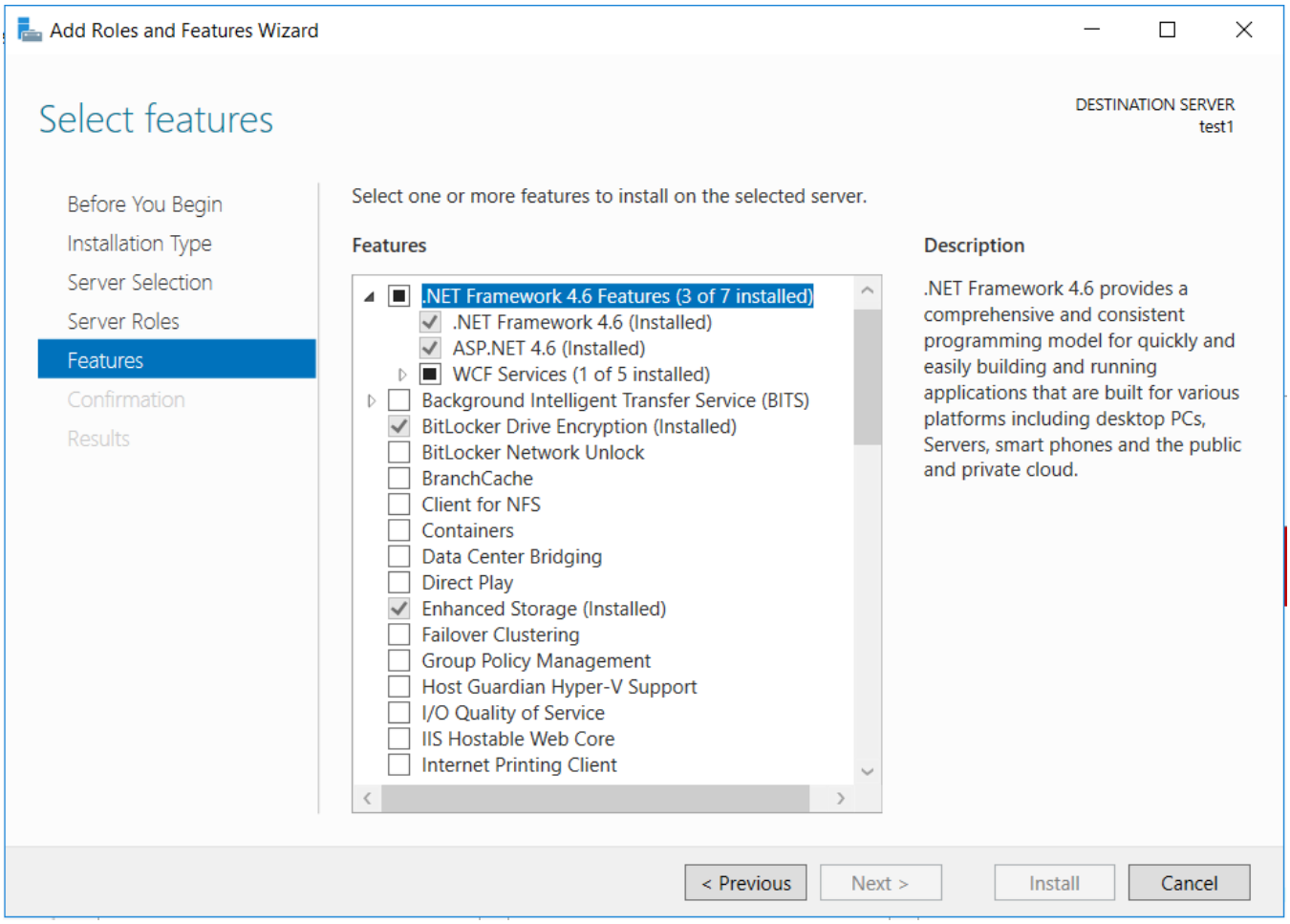

Figure 5. Screen shot of Windows 2016 Server Features selection under .NET Framework 4.6 Features that shows the selection of ASP.NET 4.

Database Setup

1. Open the command prompt.
2. Type: sqlcmd –s computer_name\sqlexpress – i path_to_script\scripts.sql, press the Enter key, and wait for the command to complete.

IIS Setup

1. Create a CancerRegistryCodingService folder under C:\Inetpub\wwwroot and copy all files and folders from the binaries folder of the repository to C:\Inetpub\wwwroot\CancerRegistryCodingService.
2. Open IIS Manager create a CancerRegistryCodingService application folder under Default Web Site, and point the physical location to C:\Inetpub\wwwroot\ CancerRegistryCodingService

Figure 6 below shows the completion of the IIS setup items described above.

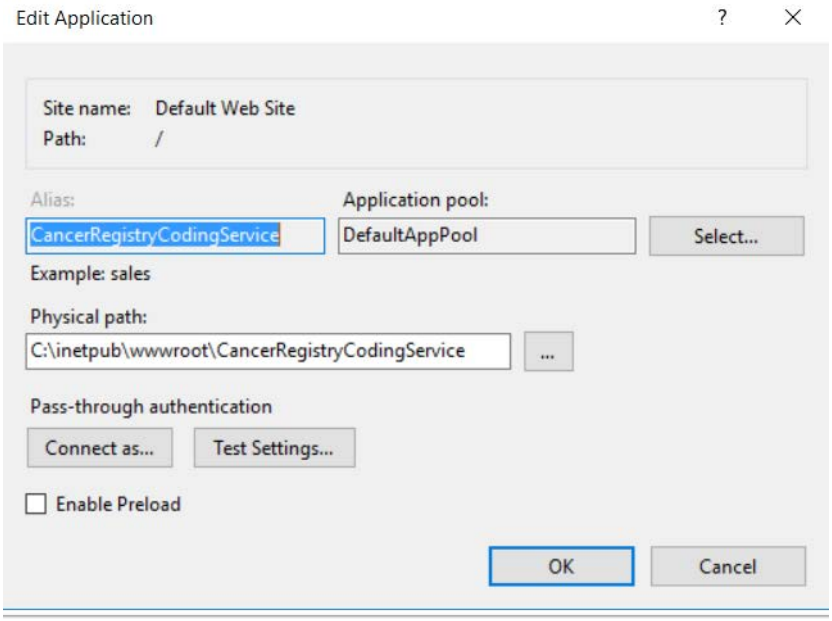

Figure 6. Screen shot of IIS Manager settings under Default Web Site.

Configure Connection to Database

Open the web.config file in C:\Inetpub\wwwroot\ CancerRegistryCodingService, and change the database connection string in the following line to point to the SQLExpress on this server.

<add name="eMaRCPlus"
connectionString="Server=computer_name1\SQLEXPRESS;Integrated Security=true; Initial Catalog=CancerRegistryCoding; MultipleActiveResultSets=true;"
providerName="System.Data.SqlClient" />

Figure 7 below shows the Connections screen after the database has been connected with the client server.

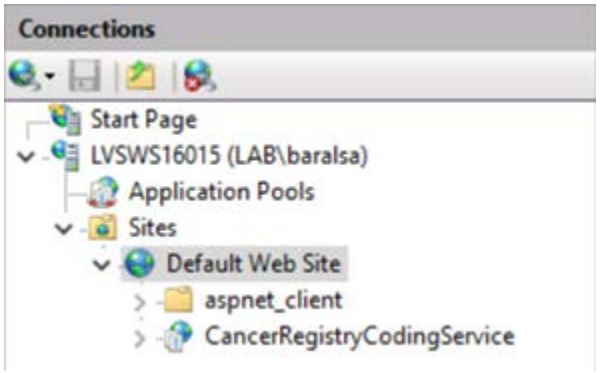

Figure 7. Screen shot of the Connections screen showing the database is connected to the server.

Feature Library

The Feature Library takes the folder of text or a .ANN extension file and generates a .BIO extension file with the features selected.

The following Python packages are required:

- lxml
- requests
- zeep
- bottle
- corenlp
- nltk
- zipfile
- pydot

All source code is provided at https://github.com/CDCgov/NLPWorkbench under HLAFeatureLibrary.

Before installing the Feature Library, install the CER Pipeline. See the CER Server section above for instructions.

1. Add the path to bin of the graphvix-2.38 into the PATH environment variable. In Python, this can be done using the following command. Change the path as needed.

```
os.environ["PATH"] += os.pathsep + '/home/jenny/CLEW_Feature_Library/graphviz-2.38/release/bin'
```

2. Install the Feature Library on the server and the web page. Add the following configuration settings to the /etc/apache2/conf-available/httpd.conf configuration file, changing the path to the wsgi file as needed:

```
# Configure server
# Server Address: http://hostname.com/clew_feature_library
WSGIDaemonProcess feature_library home=/home/jenny/CLEW_Feature_Library
WSGIScriptAlias /clew_feature_library
/home/jenny/CLEW_Feature_Library/feature_library_service.wsgi
<Directory /home/jenny/CLEW_Feature_Library>
   WSGIProcessGroup feature_library
   WSGIApplicationGroup %{GLOBAL}
   Require all granted
</Directory>
```

3. Change the server URL in the Demo\FeatureLibrary_Caller.py file, lines 18 and 45.

```
# Configure web page
```

```
# http://hostname.com/clew/feature_library
WSGIDaemonProcess feature_library_demo
home=/home/jenny/CLEW_Feature_Library/Demo
WSGIScriptAlias /clew
/home/jenny/CLEW_Feature_Library/Demo/Feature_Library_Demo.wsgi
<Directory /home/jenny/CLEW_Feature_Library/Demo>
   WSGIProcessGroup feature_library_demo
   WSGIApplicationGroup %{GLOBAL}
   Require all granted
</Directory>
```

The feature library is available at http://hostname.com/clew/feature_library/main. A screen shot of the feature library is shown below in Figure 8.

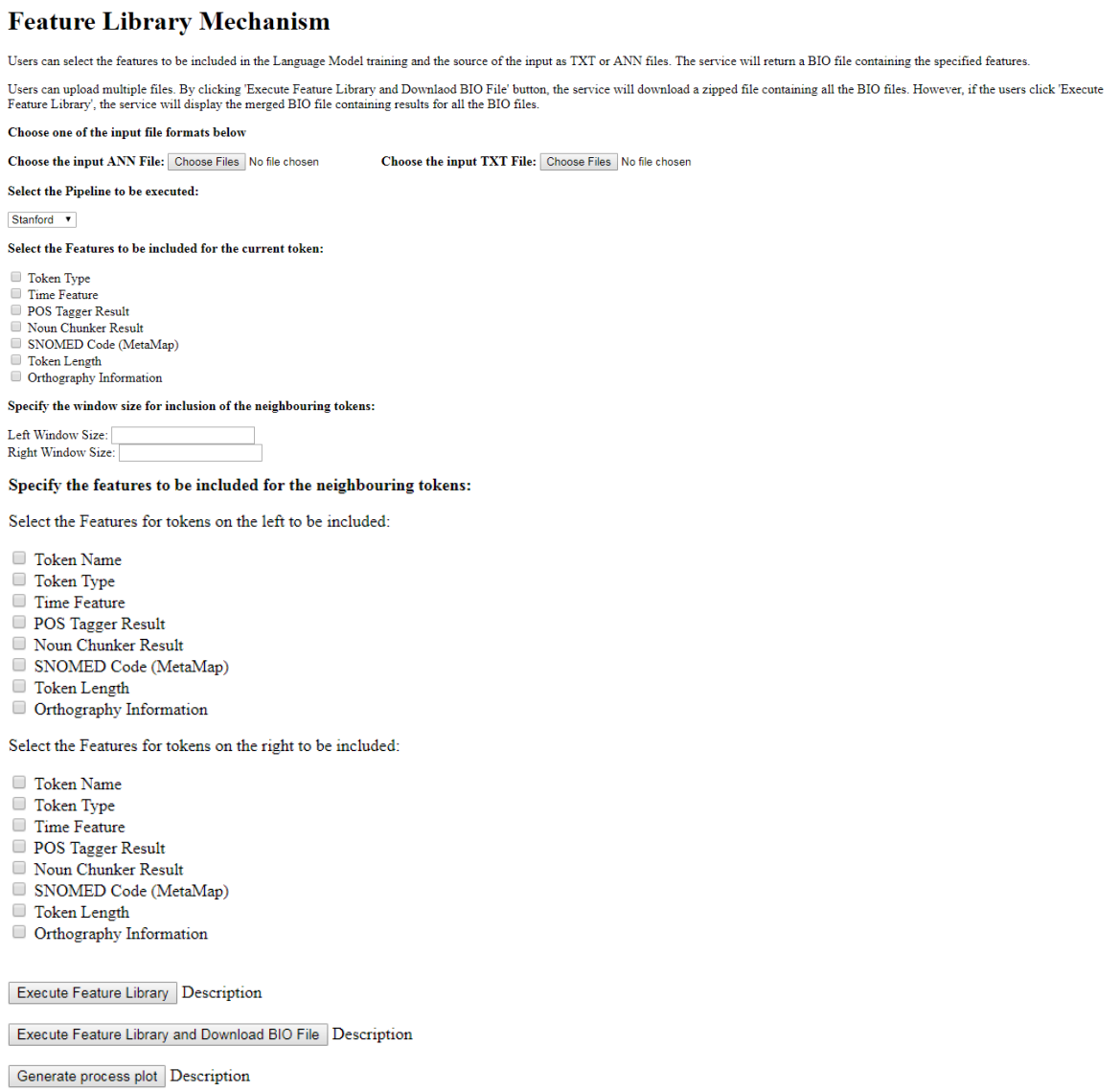

Figure 8. Screen shot of the Feature Library webpage.

Issues and Possible Solutions

Known Issues

Below is a summary of the issues identified with CDC's services.

Stanford Service and OpenNLP Service

- Tested externally via proxy for sending 500 requests. The POST requests took 10 minutes to respond for all requests. The users waited for about 5 minutes. The GET requests took 10 minutes to respond for all requests.
- Both external services and official Stanford and OpenNLP package methods are implemented to process the tokenizer, sentence splitter, and POS Tagger to reduce the chance of getting an error. If calling the external services fails, use the other method.
- Potential problems:
  - Calling external services too many times at the same time cause a *ConcurrentModificationException* or *403 Forbidden error*.
  - Using the official Stanford and OpenNLP package for large files may cause a memory error.

GATE Service

- Calling external services for all tokenizer, sentence splitter, POS Tagger and Chunker.
- Potential problem: Calling external services too many times at the same time will cause a *ConcurrentModificationException* or *403 Forbidden* error.

Possible Solutions on the Client Side

- If sending 500 requests fails for some reports, send the failed reports again after other requests have finished.
- Sending fewer reports at a time (50 to 100) is less likely to cause an error.