MUSES: A Pre-Syndromic Approach to Disease Surveillance
A Python Implementation

Boyuan Chen Yi Wei Daniel B. Neill

April 14, 2023

Contents

1 Introduction

This is a Python implementation of the Multidimensional Semantic Scan (MUSES), a novel method for pre-syndromic disease surveillance, to accompany the paper, “Pre-Syndromic Surveillance for Improved Detection of Emerging Public Health Threats” by Mallory Nobles, Ramona Lall, Robert W. Mathes, and Daniel B. Neill (*Science Advances*, 2022, in press). The code and documentation were written by Boyuan Chen and Yi Wei from New York University, supervised by Prof. Daniel B. Neill. This Python implementation is based on the original C code for MUSES (by Mallory Nobles and Daniel B. Neill, with additional contributions from Kenton Murray, Abhinav Maurya, and Yandong Liu).

The pipeline contains five steps:
- Preparation
- Preprocessing
- Training the background topic model
- Training the foreground topic model and spatial scanning to detect clusters
- Visualization of detected clusters (and providing user feedback to improve detection for future runs).

Each step requires a command line call. We offer detailed user instructions for each step, as well as explanations for selected parameters in this document. Code-level explanations are provided in each python script. For all the command line calls, please first go into the directory of the script in your terminal.

In this document, we give the sample commands to run MUSES on a synthetic sample dataset (which can be found in data/synthetic_data). This synthetic dataset was created using text strings drawn uniformly at random from the ICD-9 lookup table in place of actual chief complaints, with the remaining data fields drawn at random, in order to ensure that there are no patient identifiers or other personally identifiable information in the dataset. There is also a synthetic event manually injected into the data on January 30th, 2022, that you will find if you run the code with the sample command lines. The Python file that generates the synthetic data is included, but don’t look at it unless you want to spoil the surprise!

Additional, real-world Emergency Department chief complaint datasets from the NYC Department of Health and Mental Hygiene have been used to develop the software and for our paper cited above. These ED surveillance data are subject to restrictions protecting personally identifiable health information. Full data used in the paper are available to researchers through a signed Data Use Agreement with NYC DOHMH. Specific files to be shared include "NYC Emergency Department chief complaint data, January 2020-June 2020" (used for the COVID-19 analysis in our paper), and "NYC Emergency Department chief complaint data, 2010-2016" (used for the remaining analyses in our paper). Interested researchers may request access by contacting the New York City Department of Health and Mental Hygiene, Bureau of Communicable Disease, Syndromic Surveillance Unit, e-mail: nycdohed@health.nyc.gov, with cc to Robert Mathes (rmathes@health.nyc.gov).

We also encourage public health practitioners to apply these methods to data collected

from their own jurisdictions, and are happy to provide help and advice. Please contact Prof. Daniel B. Neill, e-mail: daniel.neill@nyu.edu, if you are a public health practitioner (federal, state, or local) and interested in using the MUSES software for day-to-day pre-syndromic surveillance.

For more general questions, advice, or requests, please contact the following team members:

Boyuan Chen - boyuan.chen@nyu.edu
Yi Wei - y.wei@nyu.edu
Prof. Daniel B. Neill - daniel.neill@nyu.edu

2 What Is MUSES

MUSES is the first software to offer a pre-syndromic approach to disease surveillance. While syndromic surveillance has made substantial progress over the past 15 years and has proven effective in monitoring known illnesses, it is not well suited for automated monitoring of rare or novel (previously unseen) diseases. As a result, an interdisciplinary team convened by the International Society for Disease Surveillance recently identified the need for new pre-syndromic surveillance methods that do not rely on existing syndromes or pre-defined illness categories[1].

MUSES enhances day-to-day situational awareness, can inform decisions about areas for targeted intervention, and acts as a safety net for public health surveillance, complementing existing syndromic surveillance approaches. In an emergency, MUSES provides early detection and insight into key questions such as the number of suspected cases, the detailed patient symptomology and the suspected illness or biothreat. It also arms public health officials with clinical data they can use to support policy decisions and obtain assistance from other agencies.

MUSES uses topic modeling to learn illness categories directly from the data, eliminating the need for pre-defined syndromes. Topic models are a set of algorithms that automatically summarize the content of large collections of documents by learning the main themes, or topics, contained in the documents. MUSES learns two sets of topics over emergency department chief complaints: a set of static topics over the historical data designed to capture common illnesses, and a set of emerging topics over only the most recent data that are optimized to capture any new illnesses not captured by the historical topics. MUSES then uses multidimensional scan statistics to identify clusters of cases isolated to a certain topic, hospital, and (optionally) demographic group of patients (e.g., the very young and very old, who may be more susceptible to illness).

As you use MUSES, you can provide feedback to indicate if the system has produced any items of concern. MUSES learns from this feedback which emerging patterns are likely to be most relevant to each public health user. This will enable MUSES to zoom in on relevant patterns, reduce false positives and provide users with actionable insights based on your own criteria for what is, and is not, relevant.

More precisely, the system maintains a list of syndromes identified by you as either relevant or irrelevant. The system can then report clusters corresponding to syndromes that were previously identified as relevant (“monitored syndromes”) as well as those corresponding to novel syndromes, while either ignoring or de-emphasizing those clusters corresponding to syndromes that were previously identified as irrelevant.

Additional information about the MUSES approach and technical details are available in the paper, “Pre-Syndromic Surveillance for Improved Detection of Emerging Public Health

[1] Faigen, Z., Deyneka, L., Ising, A., Neill, D.B., et al. (2015) “Cross-disciplinary consultancy to bridge public health technical needs and analytic developers: asyndromic surveillance use case.” Online Journal of Public Health Informatics 7(3): e228.

Threats" by Mallory Nobles, Ramona Lall, Robert W. Mathes, and Daniel B. Neill (*Science Advances*, 2022, in press).

3 Preparation

3.1 System Requirements

The scripts for preprocessing, topic model training and spatial scan can run on Windows, MacOS and Linux systems. However, the GUI (graphical user interface) for the visualization step can only run on Windows.

Please make sure that you have Python 3.8 or above installed. Then, go to the top code directory where there is requirements.txt, and call the following command in command prompt / terminal. It will install the necessary libraries to your python environment.

```
pip install -r requirements.txt
```

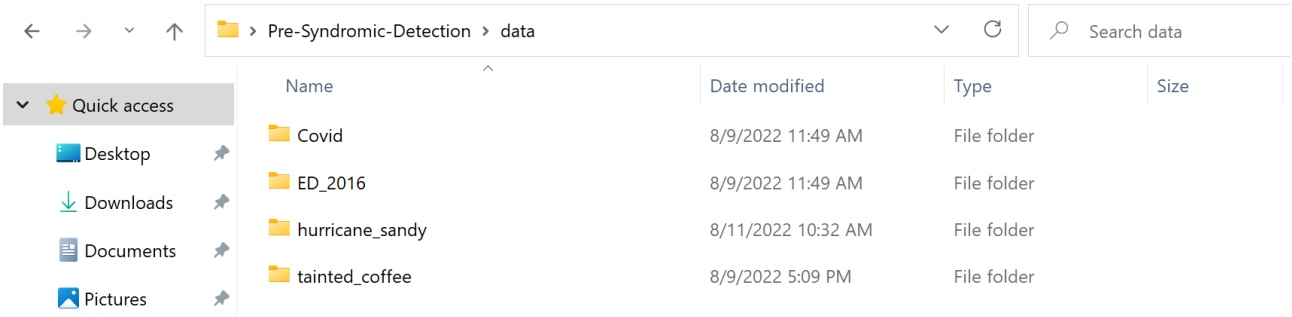
3.2 Files

Please prepare the following files:
Background data in .txt or .csv
Foreground data in .txt or .csv
Rolling age group dataframe in .csv
Hospital code dataframe in .csv

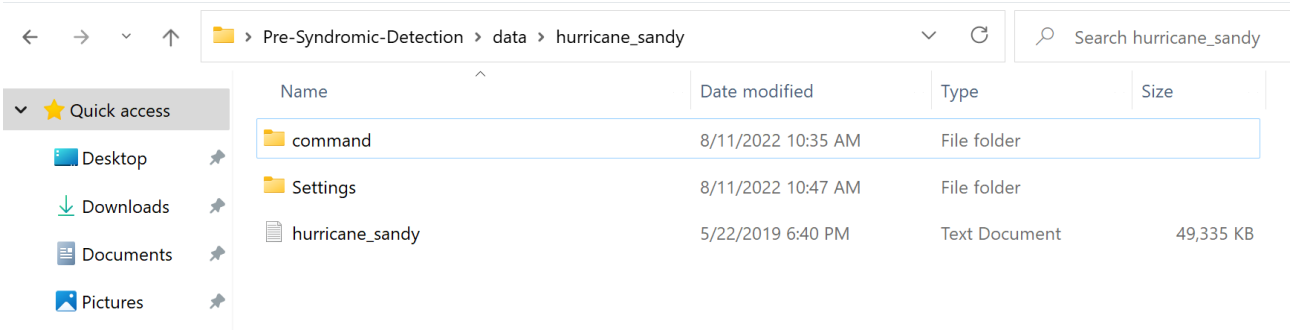
It is possible that the background data and the foreground data are in the same file, as long as you know the scope of date for the two parts, you can separate them in the preprocessing step. For the data files, create a folder in ./data, such as ./data/synthetic_data for the example case. Put the background and foreground data in the created folder.

For the latter two files, we provide sample search group files in ./preprocessing/Settings. We also provide these files in each sample data folder. You can adapt from these given samples.

We recommend that you create a data folder in the top directory to save all data, and then create a folder for each dataset inside the data folder, as shown below.



Inside each dataset folder, we recommend putting the data file on top, and put age group and hospital code in Settings folder, as shown below.


Pre-Syndromic-Detection
data
hurricane_sandy
Search hurricane_sandy
Quick access
Desktop
Downloads
Documents
Pictures
Name
Date modified
Type
Size
command
8/11/2022 10:35 AM
File folder
Settings
8/11/2022 10:47 AM
File folder
hurricane_sandy
5/22/2019 6:40 PM
Text Document
49,335 KB

4 Preprocessing

Preprocessing is the most complicated step for users, yet it is the most important step. You cannot train a useful model with wrong data. It is easy to lose track on the files generated, so please make sure that you understand all the parameters and outputs in this step.

We preprocess the raw medical records with ./preprocessing/preprocessing.py. It generates:

- A .csv dataframe that contains the raw data file's information, along with processed chief complaint text. This preprocessed file will be titled as your input for "–output_file_name", and will always contain the columns "cc", "cc_processed", "icd", "VISITID", "date", "time", "sex", "hospcode", and "agegroup" in that order. Columns "icd", "VISITID", and "sex" may be blank if they are not present in the raw data.
- A .csv dataframe of all the search groups (sg). The file will be titled as your input for "–output_sg_dict".
- A .pickle lookup dictionary that maps sg combination to sg index. The file will be titled as your input for "–output_sg_dict".
- A .txt word-to-index dictionary file that maps each word to an integer. The file will be titled as your input for "–output_word_index_dict_file".

Note that for the complete scanning task, we need a processed foreground data file and a processed background data file; we also need a search group file that saves all the search groups, a dictionary that maps hospital-and-age pair to the search group number, and a dictionary that maps each word to a number.

In order to get a word-index dictionary that covers all the words, you have two choices:

1. Process the whole dataset once (both foreground and background), and then process background and foreground without generating dictionary. So you need to call three times.
2. Process the background data, generate a dictionary, and as you process the foreground data, provide the background dictionary in "–word_index_load_file" so the new words will add on to the background words. So you need to call two times.

Either way is valid. In our example, we provide commands for the latter method.

Before running the code, you should acquire the data's date range and icd coding version, if there is a column of icd code in the data. You should also know the hospital codes that you want to scan on. Please go to the Settings folder inside each data folder and check the hospcode search group file to make sure it contains those hospitals.

Here we provide the explanations for some parameters. Parameters not included should be self-explanatory in their names.

- -sep: How the data file separates from one column to the next one. It is usually tab ("\t"), sometimes comma (",").

- -attributes_in_sequence: "Attribute" stands for the name of that column in the dataset. For instance, "date", "agegroup". Some datasets might give different titles to the columns,

and they might have different sequence; some datasets have missing columns. To make the preprocesssing code robust to different data formats, we let users manually plug in the sequence of attributes via this parameter. The attributes should pick from "cc icd VISITID time date sex agegroup hospcode", each attribute is separated by a space. If an empty string is given, then this parameter will by default be all the attributes in the previous sequence.

You must name the attributes in exactly the same way as provided. If you put in "ICD" rather than "icd", the code will throw an exception.

The following attributes are mandatory: cc, time, date, agegroup, hospcode. The following attributes are optional: icd, VISITID, sex. You may also use x to indicate that an attribute of your dataset is not one of {cc, time, date, agegroup, hospcode, icd, VISITID, sex} and thus should not be included. If your data file does not have an optional attribute, you should not include it.

Note that the number of attributes you provide to this parameter must equal the number of columns of your data file. The code will throw an exception if they are not equal. Therefore, you must faithfully reflect all attributes in your data file.

Here is an example. If your data file has the following attributes: "chief_complaints, Discharge_icd, day, time, age, ignore_me, HOSP_CODE", then you should set this parameter to: "cc icd date time agegroup x hospcode". Again, "x" is used to ignore the attribute, "ignore_me". Note that in this example there are no VISITID and sex attributes, so we will not include them. The output file will have all the default attributes. Those that were not included in the input data will be filled in with empty strings.

- -search_group_attributes: The attributes that together determine a search group. Please always keep it as "hospcode agegroup".

- -icd_map_file: The chief complaint text alone is often very short, and the icd code usually contains information that is not included in the text. Therefore, we map the icd code to the text description and concatenate with the chief complaint text. This way, the corpus for topic modeling is longer, thus more likely to generate better results.

The preprocessing script converts icd code to text and concatenates it to the chief complaints. Please check the icd version (either 9 or 10) of the data and use the corresponding txt file. Usually, data after 2016 should use icd-10; data before 2015 should use icd-9.

- -chunksize: There might not be enough memory to process the whole dataset at once, so we process and write the file once a chunksize of data. You can adjust the chunksize in the call. Default is 100000.

- -correcting_misspell: a list of misspelled word to correct word mapping. For example, "abdomianl" is changed to "abdominal". You can always open the file and add on to it, as you detect more misspells. It is currently in alphabetical format, but you don't have to keep it that way. It works the same when it is shuffled. But please make sure there is no repetitive words on the left, as the dictionary will throw error.

- -remove_word_list: a list of "useless" words that distracts users and pollutes the dataset for training. Examples are, "weeks", "of", "because", "mother", "this". Any common word that is not related to syndromes can be removed. You can always open the file and add on to it.

- -output_all_search_groups: The output file path that saves all the search groups we want to scan. Each "search group" represents a combination of hospital code and age range. For example, ["HOSP12","15-19"] might be represented by search group "802" (this is just an example). Eventually, this file is a one-column dataframe, where each row saves a search group - a combination of numbers connected by dashes. When set to ”, the script won’t output this file.

Now we move on to the calls on our synthetic sample dataset. First we process the background data (synthetic_data_2021.csv), which consists of the full year 2021, to get the preprocessed data (processed_2021.csv), search group dictionary (sg_full.csv and sg_dict_full.pickle), and word-to-index dictionary (word_dict_2021.txt):

```
cd "MUSES Open Source Software Code/preprocessing"
python preprocessing.py
--functionality=clean_data
--input_file=../data/synthetic_data/synthetic_data_2021.csv
--sep=","
--attributes_in_sequence="date time hospcode agegroup cc"
--search_group_attributes="hospcode agegroup"
--search_group_file_names="../data/synthetic_data/Settings/
   single_hospcode_searchgroups.csv ../data/synthetic_data/Settings/
   age_search_groups_all.csv"
--icd_map_file="./dicts/icd9_to_text_list.txt"
--start_date="01/01/2021 00:00"
--end_date="12/31/2021 23:59"
--chunksize=100000
--correcting_misspell="./dicts/correct_common_mistakes_list.txt"
--remove_word_list="./dicts/remove_word_list.txt"
--output_file_name="../data/synthetic_data/processed_2021.csv"
--output_all_search_groups="../data/synthetic_data/sg_full.csv"
--output_sg_dict="../data/synthetic_data/sg_dict_full.pickle"
--output_word_index_dict_file="../data/synthetic_data/word_dict_2021.txt"
--word_index_load_file=""
```

Then we process the foreground data (synthetic_data_Jan_2022.csv), which consists of the month of January 2022, to get the preprocessed data (processed_Jan_2022.csv) and updated word-to-index dictionary (word_dict_full.txt), starting from the previously created word-to-index dictionary (word_dict_2021.txt) but also adding any new words in the foreground data. We leave the two search group output paths blank, so we will not output these files.

```
python preprocessing.py
--functionality=clean_data
--input_file=../data/synthetic_data/synthetic_data_Jan_2022.csv
--sep=","
--attributes_in_sequence="date time hospcode agegroup cc"
--search_group_attributes="hospcode agegroup"
--search_group_file_names="../data/synthetic_data/Settings/
   single_hospcode_searchgroups.csv ../data/synthetic_data/Settings/
   age_search_groups_all.csv"
--icd_map_file="./dicts/icd9_to_text_list.txt"
```

```
--start_date="01/01/2022 00:00"
--end_date="01/31/2022 23:59"
--chunksize=100000
--correcting_misspell="./dicts/correct_common_mistakes_list.txt"
--remove_word_list="./dicts/remove_word_list.txt"
--output_file_name="../data/synthetic_data/processed_Jan_2022.csv"
--output_all_search_groups=""
--output_sg_dict=""
--output_word_index_dict_file="../data/synthetic_data/word_dict_full.txt"
--word_index_load_file="../data/synthetic_data/word_dict_2021.txt"
```

5 Train Static Topic Model on Background Data

We train the static model with the classic algorithm - Latent Dirichlet Allocation (LDA). You can view the original paper here[2]. In general, our goal is to learn a topic-word distribution. We call this distribution Phi (Φ).

We now train the background topic model with K (num_static_topics) topics on then background dataset, with ../topic_modeling/train_background.py. Please change the directory to the topic_modeling folder.

This script will output a .npz file saving Phi, the topic-word distribution of K background topics. Phi is of shape [K, number_of_words]. Each sub-array Phi[k] sums to 1. This array will be used in foreground training. Note that you can also get the background topics from monitored caselines in the GUI. To see how it works, please refer to the last step.

Here, "verbose" means every how many epochs would you like the computer to tell you it has finished. At each verbose print, you will also see the time lapse between the current print and the previous print. If you set it to a negative number, such as -1, the script will not print anything.

```
cd ../topic_modeling
python train_background.py
--data_file="../data/synthetic_data/processed_2021.csv"
--dict_file="../data/synthetic_data/word_dict_full.txt"
--num_static_topics=25
--static_iters=3000
--verbose=100
--checkpoint="../data/synthetic_data/checkpoint/static_2021_25topics.npz"
```

This script will not take very long (typically, a couple of minutes) since the synthetic dataset is relatively small. Nonetheless, in real practice, this step might take several hours, depending on the size of the background data.

[2] https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf?ref=https://githubhelp.com

6 View Trained Topics

At this point, you can intuitively view the learned static topics with topic_modeling/view_topics.py. Note that the learned word-topic distributions are not directly understandable by humans, as the words are saved in an encoded format. However, we can run the script with the word_index dictionary and read its printout.

The script prints out the top [display_words] number of words and their weights in each topic. If a topic is valid and relevant, the words should be consistent in describing the symptoms of a illness. Below is the code for evaluating the static topics for the synthetic sample data.

```
python view_topics.py
--checkpoint="../data/synthetic_data/checkpoint/static_2021_25topics.npz"
--word_dict="../data/synthetic_data/word_dict_full.txt"
--display_words=10
```

A sample output for one topic could look like:

	word	weight
2519	sp	0.117470
232	back	0.066265
1457	knee	0.060241
972	fall	0.057229
2429	shoulder	0.047591
2679	swelling	0.040964
1749	neck	0.039759
70	ago	0.027109
2923	upper	0.025904
1554	lower	0.022289

The left-most column is the index of each word in the word-index dictionary.

7 Foreground Training and Spatial Scanning

We incorporate foreground topic model training, using a novel contrastive topic model, and spatial scan. We will first introduce how it works, and then move on to how to call it.

7.1 How it works

On the foreground, we look at a time window of data each time. The window slides through the range of time for the scan. We recommend that you set it to 3 hours. For example, in our synthetic_data example, we start from 1/29/2022 and end on 1/31/2022. The first 3-hour window will be from 1/28 22:00 through 1/29 00:59, the second will be from 1/28 23:00 through 1/29 01:59, and so on until the last 3-hour window from 1/31 21:00 through 1/31 23:59. The baseline range is set to 21 days in this case, so the baseline data ranges from 1/8 to 1/28.

For topic training, we first train a static LDA model on K' (num_foreground_topics) number of topics on the window data. Then we use contrastive learning to combine the K static topics with the K' foreground topics. Note that the static topics will not change, but the foreground topics will be trained to separate from the static topics. The training will happen separately for each 3-hour time window.

With the trained model, we assign a topic to each caseline in the window data and the baseline data. Then, we look at the caselines in foreground and monitored static topics. We look through each search group and calculate the likelihood ratio statistic score $F(S)$ (view the calculation in the paper). To avoid division by zero, when $B < 0.5$, we set $B = 0.5$. If at least one search group exceeds the threshold score, we save the cluster of the search group with the highest score. Then, we move on to the next window.

The script will generate a file that contains all the caselines of all the detected clusters. It will then process this file to generate files for visualization.

7.2 Output and Command Calls

Running topic_modeling/semantic_scan.py generates a collection of caselines. Each caseline contains multiple rows of reports from the same topic. They are "emerging" cases as their occurrence in the foreground is high compared to the background. The scan goes window-by-window. Each time we look at a time window of several hours, and we slide the window through the foreground data. In each window, we consider all the search groups and all the topics.

The output will be:

novel_raw.csv: A raw caseline file that contains all caselines of all clusters in emerging topics (foreground topics). This file is generated first and is used to generate the latter output files starting with "novel".

monitored_raw.csv: A raw caseline file that contains all caselines of all clusters in monitored topics (the static topics you imported from the GUI, starting with "1"). This file is generated first and is used to generate the latter output files starting with "monitored".

novel_caselines.txt,
novel_cluster_summary.txt,
novel_topicwords.txt.
monitored_caselines.txt,
monitored_cluster_summary.txt,
monitored_topicwords.txt.

The static topics can be read either from a static checkpoint gained from background training, or from monitoring in the GUI, or both. You should provide at least one of them. If you provide both, the script will concatenate the two topic files with the monitored topics going first. The monitored topics are generated by users' selections in the GUI. Note that normal static topics will not be scanned - only the monitored topics and the newly-learned foreground topics will be scanned.

After the contrastive topic model is trained, we will assign a topic to each caseline with the trained parameters using a method similar to expectation maximization (EM). You can check the details of this method in the paper. It is an iterative function, and we set the number of iterations to 50 by default. It is not an adjustable parameter.

Below we provide the command call for importing static topics only.

- -static_iters: How many iterations to train the normal LDA model on window data.

- -contrastive_iters: How many iterations to train the contrastive LDA model on window data.

- -verbose: If set to True, the terminal will print the current time window in real-time.

- -step_size: How many hours should the window skip to move on to the next window. We usually set it to 1.

- -window_size: How many hours should a window contain. We usually set it to 3.

- -baseline_size: How many days previous to the current window should be considered to calculate for baseline (B). We usually set it to 21 or 28.

- -score_threshold: Only take clusters with F(S) above this score. Note that increasing score_threshold will typically speed up the run time, as the scan can be skipped if there are not enough cases in any emerging or monitored topic to achieve a score over the threshold.

- -cluster_word_num: How many words you want to include in GUI and future runs for each cluster. Note that some clusters might have over 20, or even 30 words, a lot of which are irrelevant and look messy on GUI's pie-chart. We therefore set this value so we can se-

lect only the top-weighted words in each cluster. We recommend setting this value to 12 or 15.

- -topic_weight: This boolean value determines how to distribute weights for words in detected clusters. If set to true, a word's weight will be [weight in assigned topic distribution] $\times$ [number of occurrences in cluster]; if set to false, the weight will be determined only by the number of occurrences.

- -cluster_dir: The output directory.

- -concatenate_agegroup: Only set to true when the contents in agegroup column is an integer range. For example, 11-15, 66-70, 95+. If set to true, in the gui, you will see each cluster's consecutive agegroups concatenated. For example, if a cluster has age range [11-15, 16-20, 21-25, 36-40, 41-45], then in the GUI it will show the age range [11-25, 36-45]. If set to false, each cluster's agegroup will simply be uniquefied and sorted.

If the data file agegroup has a different format, or not using "-" to connect the ages, or you are not sure about the format, please set this value to "False".

```
python semantic_scan.py
--full_scan_file="../data/synthetic_data/processed_Jan_2022.csv"
--dict_file="../data/synthetic_data/word_dict_full.txt"
--sg_dict="../data/synthetic_data/sg_dict_full.pickle"
--sg_list="../data/synthetic_data/sg_full.csv"
--static_checkpoint="../data/synthetic_data/checkpoint/static_2021_25topics.npz"
--import_monitored=""
--num_foreground_topics=25
--static_iters=1000
--contrastive_iters=1000
--verbose=False
--step_size=1
--window_size=3
--baseline_size=21
--score_threshold=4.0
--start_date="2022/01/29"
--end_date="2022/01/31"
--topic_weight=False
--cluster_dir="../data/synthetic_data/clusters"
--concatenate_agegroup=True
```

7.3 Generate files for GUI

Sometimes you might not want to wait until the full file is scanned. For instance, if you call the script to scan from 03/01 to 04/30, and you stop the terminal at 03/31, then you will have one month of data saved. However, since semantic_scan.py generates raw caselines first (novel_raw.csv and monitored_raw.csv) and then generates the other files, rather than simultaneously, you will not get the other files, which you will need for visualization in the next step. On the other hand, the two raw files have valid cluster information until 03/31. Therefore, we provide an independent script just to generate the other files from the raw caseline files. You can call it as follows:

```
python process_clusters.py
```

```
--cluster_dir="../data/synthetic_data/clusters"
--concatenate_agegroup=True
```

8 Visualization (GUI)

Our graphical visualization interface (visualize.py) visualizes the clusters produced by MUSES and enables creation of “monitored” and “ignored” static topics to be used in future scans. To use the GUI, simply go into the directory where visualize.py is and call the script with two required command line parameters, results_folder and monitored_topic_file. For example:

```
cd ../visualization
python visualize.py
--results_folder="../data/synthetic_data/clusters"
--monitored_topic_file="../data/synthetic_data/monitored_topics.csv"
```

The cluster folder should contain six .txt files created by a run of semantic_scan.py, including novel_cluster_summary.txt, novel_caselines.txt, novel_topicwords.txt, monitored_cluster_summary.txt, monitored_caselines.txt, and monitored_topicwords.txt. These files contain the data to be visualized, and are not changed by running visualize.py. If the .txt cluster files are correct, the GUI then will load and visualize the clusters.

The monitored topic file will be created if it is not already present. It can be empty or already contain topic lines of the form “1_word_probability_word_probability...” for monitored static topics or “0_word_probability_word_probability...” for ignored static topics. By choosing “Incorporate Cluster in Future Runs” in the GUI, you can append new topics to this file. This does not change the data being visualized, but can be used for future runs.

The GUI always visualize the novel monitors in the main window. To examine the monitored clusters, simply click the monitored clusters button on the top right, and a new window containing only monitored clusters will pop up. The top half of the window is a table where each row represents a single cluster. To sort the clusters, you can click the title of each column. To examine the case and symptom info of a specific cluster, click a cluster, and the cases associated with that cluster and a pie chart showing the topic words will be shown on the bottom half of the window. To sort the cases, you can also click the title of each column. The pie chart on the right represents how important a topic word is. The larger the area, the higher the probability that the word is relevant to a certain cluster.

To process the clusters, simply click the “include cluster in future runs” button to decide whether a cluster is monitored or ignored in future scans. Clicking that button will pop up a new window, in which necessary changes can be made to a word’s spelling and probability. To modify a word, you can select the word first by clicking it and then click the edit button to change the spelling and/or probability. Then, you can click the check box at the bottom to monitor this cluster. If the box is checked, the cluster will be labeled as monitored and shown in future runs when incorporated. To incorporate the cluster/syndrome, simply click the incorporate syndrome button at the bottom. This will append the syndrome to the monitored topics file, which can then be passed as an input, via the command-line parameter “import_monitored”, into semantic_scan.py.

9 Merging Clusters

You may want to merge clusters that have some degree of overlap. Merging clusters is an optional post-processing step. It can help consolidate duplicate information or highly similar clusters. The module compares the clusters in the raw caselines and calculates their similarity score. If the score exceeds the similarity threshold then the clusters are merged. After each merge, the algorithm will continue comparisons until it exhausts all potential matches. It will run on both novel and monitored clusters if available. After the merged caselines are outputted, process_clusters will be called to generate the corresponding files.

- -merge_window: The merge window is the range of time in hours that we consider potential clusters to merge.

- -similarity_threshold: This is the percentage threshold that we use to determine if the clusters are similar and should be matched. The default is set to 0.95 (95%).

- -merge_on_duplicate_case_only: This will limit potential matches to those that have cases with the exact same VisitID or all of the same case attributes (cc, time, date, and agegroup).

```
python merge_clusters.py
--merge_window=1
--similarity_threshold=0.95
--merge_on_duplicate_case_only=True
```

You may also visualize merged clusters in the same way that you can visualize the original results:

```
python visualize.py
--results_folder="../data/synthetic_data/clusters"
--visualize_novel_merged="True"
--visualize_monitored_merged="True"
--monitored_topic_file="../data/synthetic_data/monitored_topics.csv"
```

More example commands may be found in:

```
MUSES Open Source Software Code/data/command/merge_clusters.txt
```

10 Data Files Accessible Through DUA

In this section, we provide an overview of several sample datasets. However, these datasets are not public but are available to researchers through a signed Data Use Agreement with NYC Department of Health and Mental Hygiene, as described above.

We recommend that when encountering a new dataset, you acquire the same information before preprocessing. You can do so by using data_skimmer.py introduced in the next section.

A complete raw data file should have the following columns: cc, Discharge_ICD, VISITID, time, date, sex, agegroup, hospcode. The ICD, VISITID, and sex attributes are optional, while cc, time, date, agegroup, and hospcode are mandatory.

tainted_coffee_sample.txt has 366,914 rows. It's sequential in both date and time. The earliest date is 10/27/14; the latest date is 11/27/14. The hosp code is HOSP00-51 without 42. The coding is icd_9. In the sample run, we set data in 10/27-11/23 to background (303,029 rows), and 11/24-11/27 to foreground (44,054 rows).

ED_2016_only.txt has 4,460,530 rows. It's sequential in date, but not in time. The earliest date is 12/03/15, the latest date is 12/27/16. The hosp code is HOSP00-53, without 42. The coding is icd_10.

ED_2016_halfyear.txt has 2,368,376 rows. It contains half a year of data from ED_2016_only.txt. The earliest date is 06/01/16, the latest date is 12/27/16. The hosp code is HOSP00-53, without 42. The coding is icd_10. This dataset serves as the background data for hurricane_sandy scanning. In the preprocess step, we ignore HOSP50-53, since they are not included in hurricane_sandy set.

hurricane_sandy.txt has 657,938 rows. It's sequential in both date and time. The earliest date is 09/16/12; the latest date is 11/16/12. The hosp code is HOSP00-49 without 42. The coding is icd_9. In the sample run, the whole set is the foreground, and we scan the date range 10/15-11/16. The background is trained on ED_2016 data for a half-year range: 06/01-12/27.

Covid (ED_2020) has 1,592,575 rows. It is not sequential in date or time. It does not have the VISITID column. Since this set's date format is different from other datasets, we re-formatted the date and time to generate ED_2020_formatted.txt. Please use this file as the raw data instead. The earliest date is 01/01/16; the latest date is 06/30/16. The hosp code is Hosp1-53. The background is trained on 01/01-02/28. We run two separate scans: one on 03/01-04/30, one on 05/01-06/30.

11 Helper Functions

There are some helper scripts that process the data in one aspect, faster than preprocessing.py. We describe one such function below.

11.1 data_skimmer.py

This script is in ./preprocessing. It is created for users to get an intuitive view of the data before preprocessing. The reason to run this script is that .csv viewers (Excel, for example), cannot open too big files due to memory constraints. All the modes return the covered hosp code and the size of the data.

There are three skim modes. First is sequential. It skims through the file from start to end, each time showing a chunksize of caselines.

```
python data_skimmer.py
--mode="sequential"
--data_file="../data/synthetic_data/synthetic_data_Jan_2022.csv"
--sep=","
--chunksize=20
```

The random mode shows caselines randomly, each time showing a chunksize of caselines.

```
python data_skimmer.py
--mode="random"
--data_file="../data/synthetic_data/synthetic_data_Jan_2022.csv"
--sep=","
--chunksize=20
```

The target mode lets you view a specific row. You can do so by giving the row index. It shows you five rows of caselines, centered around the target caseline. This mode only shows you cc, date and time.

```
python data_skimmer.py
--mode="target"
--data_file="../data/synthetic_data/synthetic_data_Jan_2022.csv"
--sep=","
--target=500
```