KENNESAW STATE UNIVERSITY
COLLEGE OF COMPUTING AND SOFTWARE ENGINEERING
School of Data Science and Analytics

Exploratory Applications of NLPs for Developing Syndromic Surveillance Queries

Alyssa Venn
Dr. Joe DeMaio

ABSTRACT

Syndromes, or a grouping of symptoms used by hospitals and the CDC, are currently developed by hand, something that is time-consuming and not overly generalizable to all populations. There is currently no machine learning or NLP used in this space, so the goal of the project was to introduce a novel way to develop syndromes. The result is the development of an alternative methodology, specifically STM and LDA analyses, to be further explored as timely and specific methods for enhancing existing and generating new syndromes.

INTRODUCTION

Currently, the CDC's syndromes are created by hand by experts in the field. A doctor will create syndromes, or groups of symptoms, using clinically-relevant chief complaints and discharge diagnoses. This list is curated into a query that accommodates both inclusionary and exclusionary criteria. The queries are sent to other surveillance experts for validation.

Syndromes include things like "Fever," "GI," and "Neuro." Under the "Neuro" syndrome, we have subsyndromes like "AlteredMentalStatus" and "Enchephalitis," which have lists of phrases that *are* and *are not* associated with subsyndrome. This is used to create a SQL query, which categorizes CCDDs (a combination of Chief Complaint and Discharge Diagnosis) into the proper syndromes. As an example, "AlteredMentalStatus" is associated with terms like "consciousness," "mentation," "mental," and "obtundation." All the positive and negative terms are combined into one query. This method is time-consuming, and it is possible for human minds to miss key indicators, and so there is a desire for a more automated approach.

METHODS

Data Cleaning

First, I cleaned the data, specifically the CCDD variable. Stop words, like "the" and "and," were removed, as well as special characters and the phrase "
", which was used to delineate the CC and the DD. The variable was then tokenized into unigram and bigram tokens. Once the data was in the proper format, I tried three different methods for grouping the tokens.

Structural Topic Models (STM)

An STM was created to look at the CCDD column alone and grouped tokens based on similarity. The number of groups (N) must be provided, which means you must already have an idea of how many topics exist, or you must run the model many times looking for the ideal number. I created STMs with several values of N.

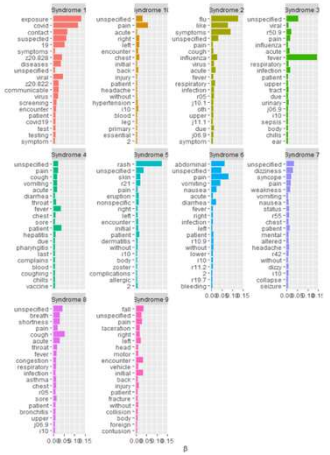
Next, I created an STM with a single group, with the data subset to only include one existing syndrome. This meant I was not creating my own syndromes, but instead taking an existing syndrome and finding the most important words for that classification.

Latent Dirichlet Allocation

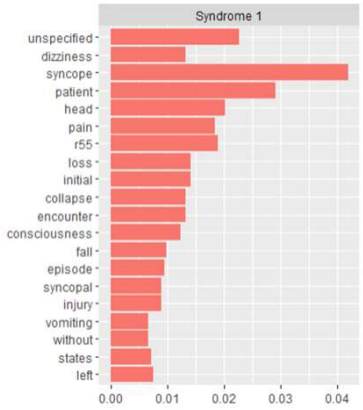
Finally, I used an LDA analysis, which assigns each token a probabilistic score of the most probable "topic" it could belong to. Unlike the STM, LDA allowed me to run multiple different numbers of topics and assigned each a coherence score, which measures the degree of semantic similarity between high scoring words in the topic.

RESULTS

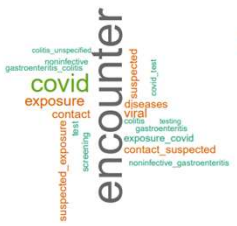
STM – Unsupervised



STM – Supervised
Syndrome - Neuro



LDA



CONCLUSIONS

- There is still more research to be done on creating syndromes with machine learning, but this has been a start. STM is not particularly applicable for multicategorical grouping but will be useful in enhancing queries for known syndrome classifications. This can potentially cut some of the human error and increase generalizability across the United States in the creation of syndrome queries.
- When it comes to the creation of new syndromes, LDA appears to be the superior methodology. More specifically, the LDA analysis doesn't require a known number of topics, meaning it has the potential to indicate when we are missing a unique syndrome. It also allows us to understand the level of importance of each token using the beta value, a log likelihood metric, which shows the importance and uniqueness of each token to a specific query.
- One difficulty in this project was the strain on the computer to run some of these analyses. With LDA, I maxed out at 40 topics, but the graph looks as though it may continue to increase in coherence past that point. In the future, I hope to use a more powerful machine and try higher values of N. Another next step is to continue the STMs on the other existing individual syndromes.
- Unsupervised methods are superior for producing new ideas, but there are two downsides. One, they are computationally heavy, taking a long time to run and requiring a powerful machine. Second, with both the LDA and unsupervised STM methods, someone still needs to interpret the topics. The analyses output lists of words, but someone still needs to figure out what they mean.

CODE

```
#LDA Model
k_list <- seq(1, 40, by = 1)
model_dir <- paste0("models_", digest::digest(vocabulary, algo = "sha1"))
if (!dir.exists(model_dir)) dir.create(model_dir)
model_list <- TmParallelApply(X = k_list, FUN = function(k){
  filename = file.path(model_dir, paste0(k, "_topics.rda"))

  if (!file.exists(filename)) {
    m <- FitLdaModel(dtm = dtm, k = k, iterations = 500)
    m$k <- k
    m$coherence <- CalcProbCoherence(phi = m$phi, dtm = dtm, M = 5)
    save(m, file = filename)
  } else {
    load(filename)
  }

  m
}, export=c("dtm", "model_dir"))
```

```
#model tuning
#choosing the best model
coherence_mat <- data.frame(k = sapply(model_list, function(x)
nrow(x$phi)),
                            coherence = sapply(model_list, function(x)
mean(x$coherence)),
                            stringsAsFactors = FALSE)
```

```
#STM
syndrome_TM_en <- stm(sparse_syndrome_en, K= 35)

#Positive beta
syndrome_topics_en <- tidy(syndrome_TM_en, matrix = "beta")

syndrome_topics_en %>%
  group_by(topic) %>%
  slice_max(beta, n=20) %>%
  ungroup() %>%
  mutate(term= fct_reorder(term, beta),
         topic = paste("Syndrome", topic)) %>%
  ggplot(aes(beta, term, fill = topic)) +
    geom_col(show.legend = FALSE) +
    facet_wrap(vars(topic), scales = "free_y") +
    labs(x = expression(beta), y= NULL)
```