# Supporting Emergency Room Decision-Making with Relevant Scientific Literature

Rohan Shetty, Liam Spurr, Didugu Sutej, and Mrigank Tiwari

## 1. Motivation and Problem Statement

Our goal in this assignment was to provide healthcare providers (HCPs) with automated access to articles that can support clinical decision-making and diagnosis. Each patient that enters the ER represents a new combination of history (including environmental factors and health history), presenting symptoms, and potential diagnoses. HCPs are already overloaded and often do not have sufficient time to keep up with the current literature regarding treatments, diseases, and the interactions between them. Our plan is to create a tool which can be used to provide these HCPs with real-time, accurate reports of research relevant to each patient who enters the hospital. By linking the intake records of ER patients to relevant PubMed articles that can provide HCPs with clinical decision support, this project aims to increase the quality of care that patients receive, while making physicians and other healthcare providers feel supported in the challenging processes of diagnosis and treatment.

There is currently no readily-available software that will link Electronic Health Records to relevant scientific articles. While there are existing tools which summarize articles and extract keywords, they do not have the capability to combine these with medical terminology to identify clinically relevant information. Our approach will identify actionable keywords in emergency room intake summaries and use these tokens to provide support in clinical decision making by delivering articles that provide information relevant to a patient's condition.

## 2. Development

### *2.1. Optimizing a medical ontology*

First, we developed a working medical ontology that would be used to automatically identify which words in the intake summaries would likely correspond to relevant medical terminology. There are only a couple of options for medical ontologies, with the most comprehensive and widely-used being Medical Subject Headings (MeSH) developed by the National Library of Medicine: (https://bioportal.bioontology.org/ontologies/MESH/?p=summary).

However, the full tree version of the MeSH was very complex and does not use standard ontology tree semantics so it would be hard to use for the purposes of this project. For this reason, we chose to instead use the .csv version provided by the National Center for Biomedical Ontology. This file was still too large for use in this project (about 250,000 terms), many of which were chemical compounds that had no relevance to this particular project. To increase the efficiency and accuracy of the project, we filtered the ontology down to a size of about 50,000 terms through trial and error of sequential filtering steps. We chose to focus on diseases (ex. diabetes), medications (ex. ibuprofen), and conditions (ex. tachycardia). The extraction of the aforementioned terms was performed in R due to its ease of use in manipulating tabular data. The code for the steps I performed is shown below.

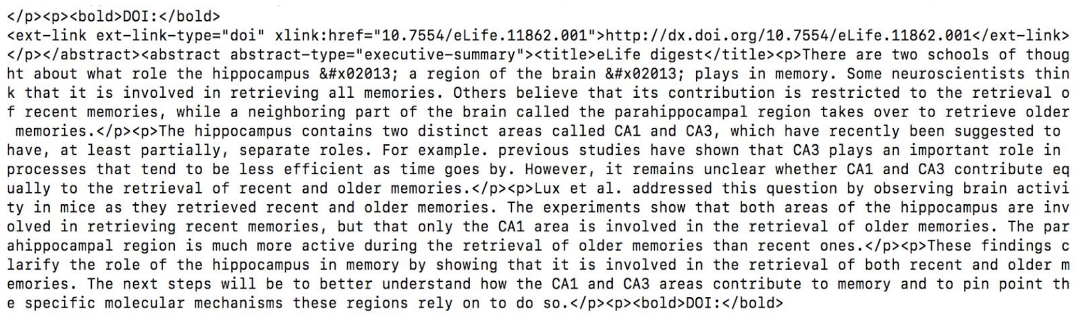
First, we removed the unnecessary columns (43) that included many NA values, as well as irrelevant information such as the source article and the definition of the term. We chose to focus on just the frequency of the terms, the term itself, and any synonyms. Then, we filtered out the proteins from the database which comprised a large part of the initial ontology. Next, we removed any terms with numbers in them. While this may have removed some useful terms (Type 2 Diabetes), it largely just removed chemical compounds (ex. hexa-1,5-diene) as upon visual inspection, many of the former type of term were written using Roman numerals. Next, we removed any terms longer than 30 characters as most diseases, conditions, and medications. After that, we removed many enzyme names, which usually end in -[a-z]ase but did not want to remove terms using the word disease, so we chose some of the most common endings. Then, we removed any words with dashes in the name, then any that end in -ol as many chemical compounds follow these patterns, while few of our desired terms do. Lastly, we eliminated any words with a frequency or three or less as they are unlikely to be commonly important terms that would be relevant to an ER patient.

## 2.2. *Preparing our data*

Our data consisted of two main components, ER intake summaries and PubMed articles, both provided by the Text Retrieval Conference (TREC) at the following link: http://trec-cds.appspot.com/2016.html#topics, which uses a small subset of the MIMIC records. According to the TREC site, the ER intake summaries contain "...admission notes from MIMIC-III…[and] describe a patient's chief complaint, relevant medical history, and any other information obtained during the first few hours of a patient's hospital stay, such as lab work… [and focus] on ICU … patients… actual data generated by clinicians… [but] contain a significant number of abbreviations as well as other linguistic jargon and style.” Before settling on this database, we first attempted to use the entire MIMIC-III database, but it proved to be far too large for our machines to handle (over 1 TB when unpacked), which would have been both too large to store and too large to perform any computations. This data was fairly easy to use because it was found in .xml files, so extracting the summaries from the larger files required only using simple regular expressions to pull out only the lines containing the summaries. Next, the tags were removed and the summaries were ready for use.

Below is an example of an extracted summary:
*A 65-year-old male presents with dyspnea, tachypnea, chest pain on inspiration, and swelling and pain in the right calf.*

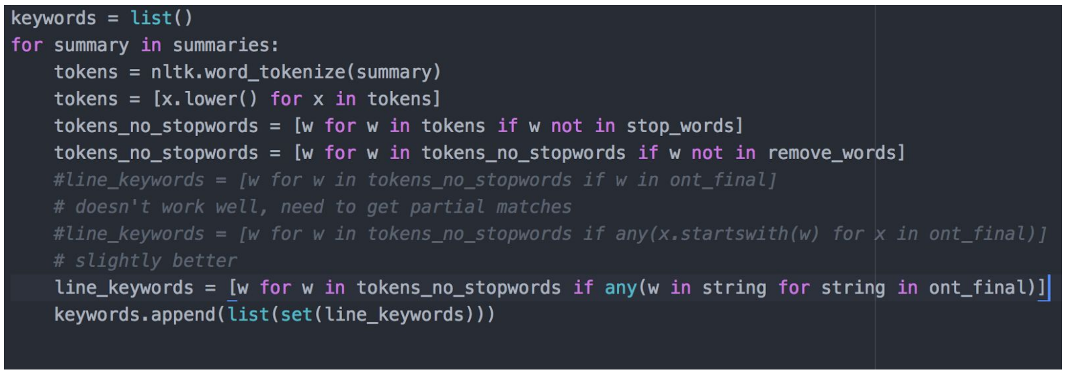
In addition, the PubMed articles needed to be similarly processed prior to summarization. However, we decided to leave the tags in and only remove the tags from the files that the algorithm identified as candidate articles, as the words in the tags should not correspond to words in the ontology and therefore not affect the results of the lookup. The tags are simple xml tags like “title”, “bold,” etc. and are not present in the shortened ontology.



```
</p><p><bold>DOI:</bold>
<ext-link ext-link-type="doi" xlink:href="10.7554/eLife.11862.001">http://dx.doi.org/10.7554/eLife.11862.001</ext-link>
</p></abstract><abstract abstract-type="executive-summary"><title>eLife digest</title><p>There are two schools of thoug
ht about what role the hippocampus &#x02013; a region of the brain &#x02013; plays in memory. Some neuroscientists thin
k that it is involved in retrieving all memories. Others believe that its contribution is restricted to the retrieval o
f recent memories, while a neighboring part of the brain called the parahippocampal region takes over to retrieve older
 memories.</p><p>The hippocampus contains two distinct areas called CA1 and CA3, which have recently been suggested to
have, at least partially, separate roles. For example. previous studies have shown that CA3 plays an important role in
processes that tend to be less efficient as time goes by. However, it remains unclear whether CA1 and CA3 contribute eq
ually to the retrieval of recent and older memories.</p><p>Lux et al. addressed this question by observing brain activi
ty in mice as they retrieved recent and older memories. The experiments show that both areas of the hippocampus are inv
olved in retrieving recent memories, but that only the CA1 area is involved in the retrieval of older memories. The par
ahippocampal region is much more active during the retrieval of older memories than recent ones.</p><p>These findings c
larify the role of the hippocampus in memory by showing that it is involved in the retrieval of both recent and older m
emories. The next steps will be to better understand how the CA1 and CA3 areas contribute to memory and to pin point th
e specific molecular mechanisms these regions rely on to do so.</p><p><bold>DOI:</bold>
```

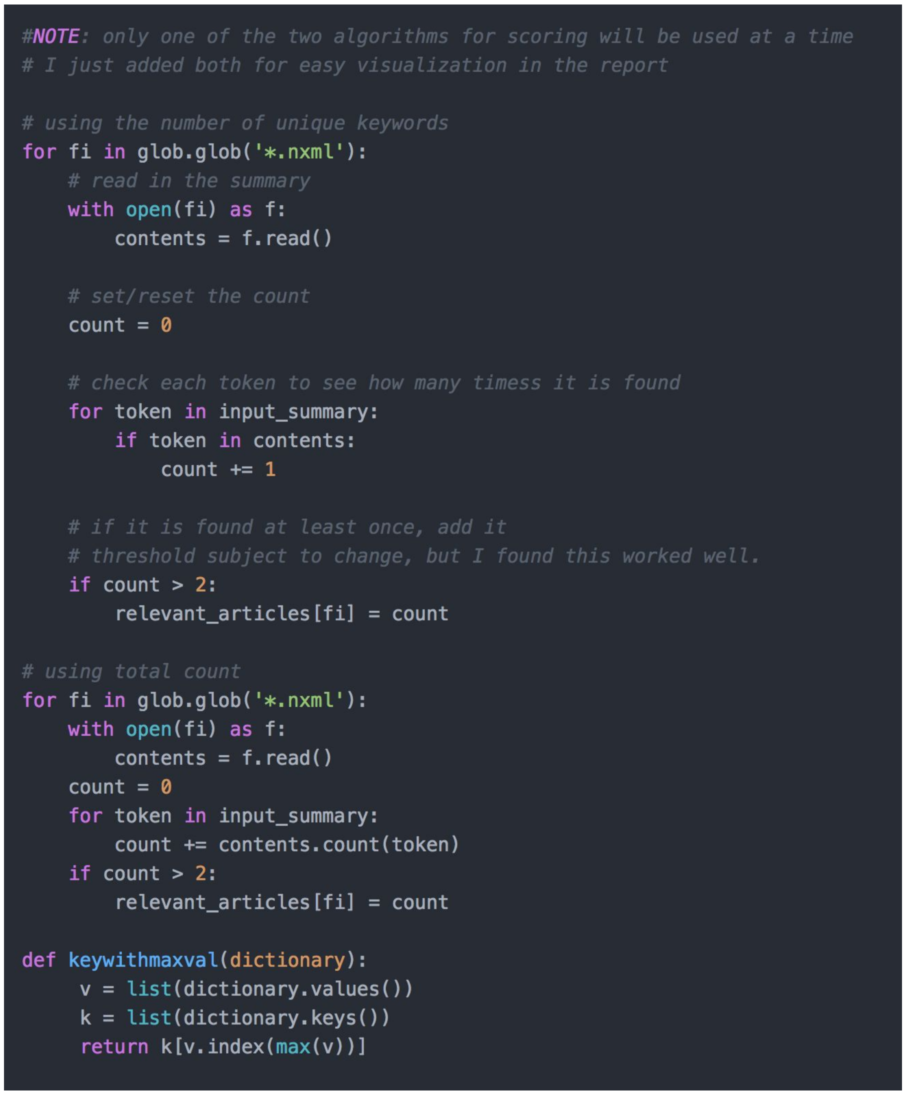
## 2.3. *Article lookup*

The next step was to use the prepared data to identify the relevant articles from the PubMed database. The first portion of this entailed extracting the keywords from the ER intake summaries. We tokenized each summary using the NLTK package and removed the words that would clearly not be important in a summary. We used the NLTK stopwords list and added onto it with other common words that seemed irrelevant for the purposes of this particular study, such as “a”, “the”, “man”, “woman”, etc. that would only serve to decrease the efficiency of the lookup and confound the results. Then, we filtered to only include words where at least part of the word was in the ontology. However, there were a couple issues here where future work would be beneficial. First, the lookup could not match correctly if the word in the ontology was not in the same form as the word in the summary. For example, the word “immunosuppression” would not be matched to the word “immunosuppressed.” In the future, it would be prudent to lemmatize or at least stem these words in order to achieve better matching rates and generate more accurate keywords for lookup. In addition, one feature that could be implemented would be falling back to synonyms if a token itself is not found. The ontology contains a column for synonyms, and while we did not implement this for this project, it could also allow for matching of words with similar meaning, even if the words do not have the same stem or lemma. Along this same line, if we were to use an ontology with a tree-structure, it would allow us to lookup related words if the initial lookup failed. In the code snippet below, the lines that contain the main keyword extraction process are shown, along with my first attempts at matching words and how we improved the matching by including partial matches and converting both the summaries and ontology to all lowercase (the ontology is converted in a different section of the code in a similar manner).



```
keywords = list()
for summary in summaries:
    tokens = nltk.word_tokenize(summary)
    tokens = [x.lower() for x in tokens]
    tokens_no_stopwords = [w for w in tokens if w not in stop_words]
    tokens_no_stopwords = [w for w in tokens_no_stopwords if w not in remove_words]
    #line_keywords = [w for w in tokens_no_stopwords if w in ont_final]
    # doesn't work well, need to get partial matches
    #line_keywords = [w for w in tokens_no_stopwords if any(x.startswith(w) for x in ont_final)]
    # slightly better
    line_keywords = [w for w in tokens_no_stopwords if any(w in string for string in ont_final)]
    keywords.append(list(set(line_keywords)))
```

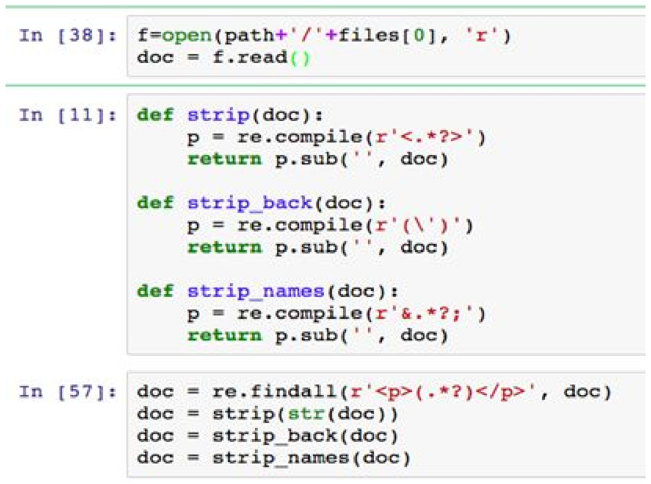
The second portion was the actual lookup of the most relevant articles using the keywords extracted in the former portion. There were several challenges that we encountered during the lookup. The first was the number of articles that my machine was able to handle. We used about 100,000 of the PubMed articles, which only comprised a small fraction of the entire database, which was too large to unpack all at once. Although this may seem like a large number of articles, the sheer diversity of topics among PubMeds 27 million articles means that it is unlikely to find any given topic among these 100,000 articles, and even less likely to find a article about a specific set of topics. By performing this lookup in a computing environment that would be able to handle this volume of data, we could improve the lookup by the sheer availability of data for the lookup process. Secondly, there were two approaches to scoring the articles in terms of relevance that we tried. First, we tried using the count of the total number of times any keyword appeared in the article. However, this was not always very effective because in many cases, one fairly irrelevant keyword would be repeated many times over in an article, which would lead it to being given a high score when it did not “deserve” it. Instead, we elected to use the approach that determined the number of the keywords that appear at least once in the article. In our testing, it performed better than the former approach but still suffered from the issue of having the keywords only play a very small role in the article, while still being present. For example, when we used the total number of words for the article in Example #2, the highest scored article (score of 1341) was about “new data on African American health professionals abroad.” We think that it would be best to play around with combining the two scores and providing relative weights to assign articles which have both a high frequency of keywords and a high number of the keywords appearing at least once the best scores. In the code snippet below, a portion of the article scoring is shown. Code for both of the scoring methods is presented, even though only one is used at a time in practice. A simple function to retrieve the article with the highest score is also shown.



```python
#NOTE: only one of the two algorithms for scoring will be used at a time
# I just added both for easy visualization in the report

# using the number of unique keywords
for fi in glob.glob('*.nxml'):
    # read in the summary
    with open(fi) as f:
        contents = f.read()

    # set/reset the count
    count = 0

    # check each token to see how many timess it is found
    for token in input_summary:
        if token in contents:
            count += 1

    # if it is found at least once, add it
    # threshold subject to change, but I found this worked well.
    if count > 2:
        relevant_articles[fi] = count

# using total count
for fi in glob.glob('*.nxml'):
    with open(fi) as f:
        contents = f.read()
    count = 0
    for token in input_summary:
        count += contents.count(token)
    if count > 2:
        relevant_articles[fi] = count

def keywithmaxval(dictionary):
     v = list(dictionary.values())
     k = list(dictionary.keys())
     return k[v.index(max(v))]
```

## 2.4. *Text extraction*

Extraction of the relevant text was performed using simple regular expressions. This was necessary because since the data set is a XML file, we had to filter out the tags and other irrelevant information. Among this unneeded content were the title and headings, so we decided to select only the text in the paragraph tag (<p>). The 're.findall()' command is used for selection of data between the paragraph tags. Since the data with in the paragraph tags had several text formatting attributes and subtitles, we had to remove all the tags in the given data to make sure we had an accurate summarization. We decided to implement this by writing functions which select the unwanted strings and replace them with blank space. The regular expression ‘<*?>’ is used to select all tags with in the paragraph tags and the ‘sub’ function is used to replace it with blank space. Similarly, we had to filter out hexadecimal code which is used to add color for specific words and we also removed unwanted special characters to obtain clean data which lead to efficient and accurate summarization. This process is split into 3 functions: *strip*, *strip_back,* and *strip_names*. These functions are displayed in the image below.



```
In [38]: f=open(path+'/'+files[0], 'r')
         doc = f.read()

In [11]: def strip(doc):
             p = re.compile(r'<.*?>')
             return p.sub('', doc)

         def strip_back(doc):
             p = re.compile(r'(\')')
             return p.sub('', doc)

         def strip_names(doc):
             p = re.compile(r'&.*?;')
             return p.sub('', doc)

In [57]: doc = re.findall(r'<p>(.*?)</p>', doc)
         doc = strip(str(doc))
         doc = strip_back(doc)
         doc = strip_names(doc)
```

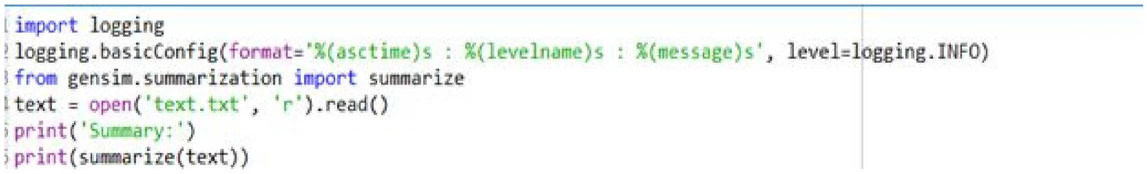
## 2.5. *Text summarization*

Text summarization is the process of shortening a text document in order to have a smaller document containing only the important points from the original document. There are two types of text summarization: extractive and abstractive. The former involves selection of existing phrases and sentences from the source document to create a new summary. The latter entails generating entirely new phrases and sentences to capture the meaning of the source document. This kind of approach is similar to the approach usually taken by humans while making a summary. This approach is suitable for supervised methods where the algorithm has previous knowledge of correct data relating to the subject.

For our project, we elected to use Gensim for text summarization. The Gensim summarizer is an extractive text summarizer based on the "TextRank" algorithm, published in article by *Mihalcea et al*. It was later improved upon by *Barrios et al.* by introducing the BM25 ranking function. We chose an extractive text summarizer like Gensim was because we did not have the labeled gold data sets needed for abstractive text summarization techniques.

The TextRank algorithm utilized by Gensim is a graph-based algorithm which identifies important vertices in a graph by taking into account global information recursively computed from the entire graph. [1] The graph used for sentence extraction is a weighted graph where each vertex denotes a sentence and the vertices are connected with weights assigned based on the relationship between two sentences based on a recommendation system, such as a sentence that refers to a particular concept and asks readers to refer to another sentence having similar concepts [1].

*Reference: Mihalcea, R., & D., Tarau. (n.d.). TextRank: Bringing Order into Texts. 1-8. Retrieved November 4, 2017, from https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf*

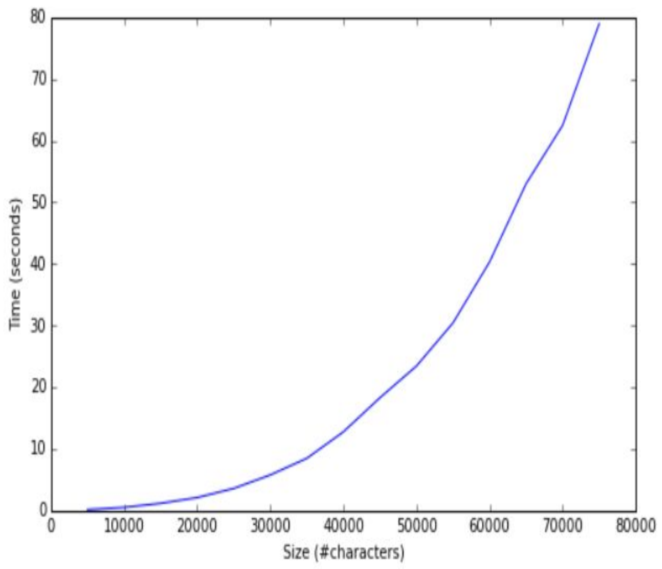


```
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
from gensim.summarization import summarize
text = open('text.txt', 'r').read()
print('Summary:')
print(summarize(text))
```

To use Gensim, we first needed to install the Gensim package in our Python environment. From Gensim, we import the summarize function and utilize it to summarize the text as shown in the above figure.

## 2.6. *Performance and drawbacks*

As described above, the data structure used in the algorithm is a graph which consists of nodes (sentences) where the weights among the edges show how the two sentences are related to each other. Because of this, every sentence will form a different graph and therefore have different running time. In general, we can say size of data set is directly proportional to the running time: as the size of dataset increases, the running time also increases. In the majority of cases, the algorithm runs in polynomial time.



Source: https://rare-technologies.com/text-summarization-with-gensim/

Unfortunately, there are still some drawbacks to using this approach. For one, Gensim does not support multithreading, which can slow performance when this type of algorithm is implemented at full-scale. In addition, as mentioned above, the performance of the algorithm can be very slow depending on the structure of a given group (worst case quadratic time). In addition, Gensim currently only works for English datasets, but in theory the TextRank algorithm can be applied to any language, as it is an unsupervised algorithm and does not require gold data for model development.

## 3. Examples and Future Work

*Example 1*

Original ER Intake Summary: *A 62-year-old immunosuppressed male with fever, cough and intranuclear inclusion bodies in bronchoalveolar lavage.*

Keywords extracted: ['bronchoalveolar', 'fever', 'cough', 'lavage', 'bodies', 'inclusion', 'intranuclear']

Except from related article summary (6 keywords found):

*Immunocompromised individuals including cancer patients, transplant recipients, and those receiving immunosuppressive therapies including monoclonal antibodies should be evaluated regularly and treated for LTBI at the time of diagnosis or just before starting immunosuppressive treatment (1, 6)., Several risk factors predispose individuals to M. To prevent reactivation of LTBI in recipients of allogeneic HSCT, it is recommended to consider administration of isoniazid (INH) prior to and post-HSCT particularly in patients living in areas that are endemic for TB (14)., M. tuberculosis infections in HSCT recipients should be based on: clinical grounds, sputum microscopy and cultures, cultures of pleural and pericardial fluid in addition to bronchoalveolar lavage (BAL) samples, bone marrow cultures, serology, molecular testing, and tissue biopsies (1, 40).*

Select keywords (from part of the summarization package): tuberculosis infections, infection, patient, therapy, immunosuppressive therapies, drug resistant, disease, sputum, cultures, pulmonary, rapidly progressive

*Example 2*
Original ER Summary: *A 65-year-old male presents with dyspnea, tachypnea, chest pain on inspiration, and swelling and pain in the right calf.*

Keywords extracted: ['right', 'dyspnea', 'pain', 'calf', 'tachypnea', 'chest', 'swelling']

Except from related article summary (6 keywords found):

*Tamponade can cause hypotension due to decreased stroke volume, jugular-venous distension due to impaired venous return to the heart, and muffled heart tones due to fluid inside the pericardium. The patient did not have visibly distended neck veins, illustrating that these findings, known as Beck's triad, are unreliable for the diagnosis of pericardial tamponade as it is seen only in a minority of patients. Dyspnea is the most common presenting finding with cardiac tamponade. Although this patient presented with normal blood pressure, it increased substantially to the patient's baseline hypertensive state after pericardiocentesis. Dialysis treatments prevented recurrences of cardiac tamponade.*

Select keywords: pulmonary, venous, blood, heart, dialysis, tamponade, albuterol, hypertensive, acute, passive, ventricular failure

The techniques described above were all implemented to produce the above outputs. Overall, Example 1 is a good example of a proof of concept of the approach we have taken. The article contains several of the main words from the ER intake summary, such as bronchoalveolar, lavage, and immunosuppression. In general, the content in the article summary seems to be relatively related to the summary itself, but not precisely so. We attempted to quantify how related the article was to the summary by looking at the keywords produced by the summarization algorithm. However, it seems that more work is needed in this area because many of the keywords produced were stopwords. For this reason, I removed those and displayed the important keywords above. In addition, while the overall theme of the article is related to the initial summary, it is important to note that the word “immunosuppressed” from the summary was not chosen as a keyword because only the word “immunosuppression” was in the ontology. Examples like this highlight the need for further development and use of lemmatization and stemming techniques as described above in the model development section. Along the same lines, it may be even more effective to use another ontology with a tree structure where the matching algorithm can fall back to words that have a similar meaning or come from the same root. Overall, this represents a positive example of an output that our model could produce while highlighting areas that must be improved before this approach is used in a professional setting.

On the other hand, Example 2 is an example of an inaccurate article. The article has almost nothing to do with the summary that was presented, but was scored highly because the keywords such as “dyspnea” appeared in the article, albeit infrequently. For this reason, it would be important to use a hybrid score that takes into account both the total frequency of keywords and the number of unique keywords present to ensure that the articles identified truly relate strongly to the summary. Moreover, providing a bigger pool of articles to pull from will also intuitively improve performance because there will simply be more articles related to a given topic. Further exploring additional summarization techniques could be another avenue of improvement. It is important that the summaries are easy to read, concise and accurate. For instance, we tried playing with the length of the summaries but were unable to settle on a length that worked best for all articles as some articles were much longer than others and it takes the length argument on a percentage scale.

All in all, we believe that since we have been able to produce some promising initial outputs, implementing this at a larger scale with the aforementioned tweaks to the lookup and summarization portions along with the full set of PubMed articles, would enable us to successfully identify articles that would fulfill our goal of supporting clinical decision making in an emergency room setting.