# PubMiner: An Interactive Tool for Demographic-enriched PubMed Searches

Joe Bockskopf
Yuning Chen
David Dowey
Bin Gao
Alberto Garza
Ian Smith

*Harvard Extension School, Harvard University, Cambridge, MA 02138*

# Table of Contents

# PubMiner: An Interactive Tool for Demographic-enriched PubMed Searches

Joe Bockskopf, jbockskopf@g.harvard.edu
Yuning Chen, yuc995@g.harvard.edu
David Dowey, ddowey@g.harvard.edu
Bin Gao, bingao@g.harvard.edu
Alberto Garza, albertogarza22@gmail.com
Ian Smith, ims864@g.harvard.edu

*Harvard Extension School, Harvard University, Cambridge, MA 02138*

## Abstract

Over one million medical research publications are added to PubMed every year, presenting an ever-increasing challenge to clinicians in finding the publications that are most relevant to their research or practice. At present, PubMed's search does not provide tools to search clinical trial articles for their target population groups, so a time-consuming manual review is required.

PubMiner, a novel open-source interactive search tool, solves this problem by augmenting PubMed search results on randomized clinical trials with extracted demographic data (sentences and tables containing demographic data). It leverages text mining and machine learning approaches to extract clinical trial participants' demographic data from the articles freely available in PubMed Central, an archive of biomedical and life sciences journal literature. New articles are processed and their related demographic data is stored in a database so that PubMiner can integrate the demographics with PubMed search results on request. The interface significantly reduces the time that clinicians spend searching for and reviewing demographic data on relevant clinical trial articles. PubMiner can guide both patient stratification for future trials and help clinicians provide the most personalized treatments possible.

**Keywords**: PubMed, PMC, Table Mining, Text Mining, NLP

[**Submission**: primary journal - Journal of the American Medical Informatics Association, alternate - International Journal of Medical Informatics]

## Introduction

PubMed is a publicly available search engine used to access the MEDLINE (Medical Literature Analysis and Retrieval System Online) database of articles as well as other various sources on biomedical topics. Currently there are over 28 million records indexed to search through PubMed. Many of these records contain links to full text articles, some of which are freely available in PubMed Central (PMC), an archive with publicly available full-text articles. For some medical researchers, this is often the primary resource leveraged to find literature relevant to their current research topic.

A key type of article describes a clinical trial. These publications are the result of experimental clinical research designed to answer specific questions about a treatment or intervention (eg., vaccines, dietary choices, medical devices, etc.). These sorts of clinical trials are conducted on volunteers, starting with smaller pilot studies and progressively expanding to larger comparative studies.

With the general advancement of medical research and the ever increasing focus on personalized medicine, a growing focus of researchers is to find clinical trials that include patients with a similar profile to their demographic segment of interest. PubMed search currently includes some advanced features, such as Medical Subject Heading (MeSH) term tagging, which are a curated set of descriptors that are arranged in a hierarchical structure, in addition to standard functions like  boolean operators, synonyms searching, and other search-enhancing functionality. However, there is currently no facility for searching for specific demographics within articles.

PubMiner is a tool that aims to add demographic-focused search capabilities to the current PubMed search function. PubMiner consists of (1) a process through which clinical trial articles' text is processed in order to extract both sentences and tables containing demographic information describing the volunteers/patients in the trial, and (2) a web application through which users can submit PubMed search queries and retrieve enhanced results that will include the extracted demographic sentences and tables.

The rest of this paper is structured with the following sections: Literature Review will discuss work that has been done that is relevant to both the identification of clinical trial articles as well as the text mining of those articles, Table Mining and Sentence Mining will discuss our methods for extracting tables and sentences from articles, and Results and Future Work will discuss the results of these methods as well as what future work can be done to improve and further this work.

## Literature Review

The idea of enhancing PubMed searches is not a new one. A study conducted almost a decade ago [Lu 2011] found 28 various PubMed-derived systems providing features ranging from enhanced ranking search; clustering into topics; extraction and displaying semantics; and improvements to the interface and retrieval experience. Techniques for text mining and table mining of the full-text content of scientific articles have been the subject of a broad body of research. Additionally, the articles in PubMed have been used for research into clustering, classification, extraction of semantic structures and relationships, and the detection of tables and their structure.

In an earlier research paper on full-text data mining of PubMed [Xu, Garten 2006], text classification was applied to separate the abstracts of randomized clinical trials into five sections: Background, Objective, Methods, Results, Conclusions. A Hidden Markov Model was used to extract the specific sentences containing demographic information.

Statistical analysis methods can be applied to perform sentence similarity analysis. Research in this vein has been conducted on the similarity of short sentence from PubMed articles [Li, McLean 2006]. In this research, semantic similarity of two sentences was calculated using information from a structured lexical database and from corpus statistics.

A major challenge in full-text data mining is to extract the relationship from the context. The techniques winning the highest F1 score in the 2010 i2b2 Challenge have been disclosed for the relation-extraction task [Rink, Harabagiu 2011]. They indicate the extreme importance of lexical and contextual features on relation extraction from medical texts.

A hybrid approach has been used to implement a method to discover, download, and extract context automatically [Tchouaa, Chard 2016]. In this research, during the extraction period, semi-experts helped to extract specific scientific facts.

Extensive studies have been conducted on machine comprehension, from answer-generation for comprehension [Tan, Wei, 2017] to solving math word problems [Wang, et al. 2017]. These efforts may be transferred to other areas; however, the lack of annotated training data has typically limited their application to PubMed data. An alternative has been suggested [Ardehaly, Culotta, 2017]: introducing a deep neural network to learn the Label Proportion (LLP) setting. Their training set contained bags of unlabeled instances associated with a label distribution for the bags.

Tables are a fundamental means of presenting data and reporting results, and are therefore commonly used in scientific articles. The information stored in tables is typically semi-structured, with few rigid rules and thus various models have been applied to tables, including lists, grids, trees, hierarchies and graphs. According to Hurst, the table mining process consists of four steps: table detection, in which existing tables are identified within a document; functional analysis, which is the detection and marking of functional areas of tables (e.g. headers, stubs, cells); structural analysis, which determines the relationship between cells; and semantic analysis, which determines the meaning of data. One conceptual approach is to segregate  tables into five key components: graphical, physical, functional, structural, and semantic,  [Hurst 2000].

Decision Tree and SVM have been used to detect tables [Hu, Wang 2002]. A conditional random forest model has been employed to extract data from tables and suggested better results compared with that from Hidden Markov Model [Pinto, McCallum 2003].

Spanned rows and columns are a big challenge for effective table mining. Using attribute value pairs on standardized tables [Sale, Chawan, 2012] gave decent results.

Since some tables are scanned, extracting data from the tables in the scanned document has also been  explored [Stadermann, Symons, 2013].

A very comprehensive study of mining tables in PubMed was conducted by Nikola Milošević for his PhD thesis [Milošević 2017]. He studied the methodology and produced a software implementation to extract some information from the tables in PubMed [Milošević, 2016, 2017]. In his research, the four steps were studied and implemented on selected files.

An attractive research on using deep learning to understand the semantic structures of tables was conducted by Kyosuke Nishida [Nishida, Sadamitsu 2017]. In this research, a deep learning network consisting a RNN layer, a CNN layer, and a classifier was used. The RNN encoded the sequence of tokens for each cell to a 3D image data, and the CNN decoded this image to capture semantic features. In the end, the classifier gave the type of the table.

Some attempts to combine the full-text mining and table mining have also been conducted. In [Govindaraju, Zhang 2013] the F1 score, which is a measure of accuracy that takes both the ability to correctly identify and the percentage of correct identifications, was doubled in a model with joint probabilistic inference combining tabular and context analysis together. Another tool that can be used for general dark data mining, regardless of context, table, figure, etc., is DeepDive [Zhang, Shin 2017].

## Identifying Randomized Clinical Trials

### Narrowing PubMed search to Randomized Clinical Trials (RCT)

RCTs are of high interest to medical researchers. While there is a PubMed tool to identify types of “Clinical Queries”, this service does not constitute a filtering mechanism, and does not yield a full set of RCT articles. This is due to both RCTs being a subset of what falls into the ‘Clinical Queries’ category and inherently inaccurate tagging. Moreover, there is a Clinical Trial publication type in PubMed, yet this cannot be used to obtain a definitive set of articles either since not all RCTs have been indexed with it. Using this Publication Type would yield a higher precision for the search, by eliminating many non-RCTS, but it would also risk not including RCTs of interest. Another possible alternative would be to use the Cochrane CENTRAL database of RCT reports to perform a simple search, as in [Royle, Milne 2003] and [Royle, Waugh 2005]. But while this database makes substantial progress in classifying RCTs, and could improve the specificity of the search, as with the search by publication type, not all RCTs have been classified.

The first step in identifying the broad corpus of PubMed articles with the highest likelihood of finding RCTs is to provide a refined set of search filter terms to the PubMed E-utilities API [Entrez Programming Utilities Help]. In a comparative study of 38 search terms [McGibbon et al. 2009] it was found that simply using `randomized+controlled+trial[pt]` yielded a sensitivity of 93.7% on Medline searches. In our work, an emphasis was placed on sensitivity, which is the ability to detect RCTs, i.e. a goal of the most number of true positives rather than the least number of false positives. This was because subsequent Table and Sentence Mining would occur later in the process and would help to *de facto* identify RCT articles. Additional search terms were added with the aim of improving precision, which is the percentage of predictions that are correctly identified as RCTs, while maintaining high sensitivity. Figure 1 contains the terms used through the E-utilities API (eutils) to identify the PMIDs of RCTs to be used as the broad corpus of articles.

```
((randomized+controlled+trial[pt])+OR+(controlled+clinical+trial[pt])
+OR+(randomized[tiab]+OR+randomised[tiab])+OR+(placebo[tiab])+OR+(dru
g+therapy[sh])+OR+(randomly[tiab])+OR+(trial[tiab])+OR+(groups[tiab])
)+NOT+(animals[mh]+NOT+humans[mh])
```

*Figure 1. Randomized control trial filter*

The terms have been referred to as the Cochrane Highly Sensitive Search Strategies [the Cochrane Work website] for identifying randomized trials in Medline, but are extensible to PubMed searches. The query should exclude animal trials, which is of particular importance. Of note is that it does not use the MeSH terms for Publication Types of “Clinical Trial”, “Clinical Study” or “Clinical Trial, Phase X” , since many articles have not been indexed by these MeSH terms as not all the articles available in PubMed have been classified by Publication Type. Furthermore, filtering by Article Type is not possible in PubMed Central, which would indicate that many of the articles that were used to populate our database are not, in fact, indexed. The search terms that were used have also been proposed as an optimal search strategy in [Dickerson et al. 1994] and [Glanville et al. 2006].

From this broad corpus, further refinement is performed to narrow down to target articles where the token stem appears in the title or abstract of the article. This step would improve the sensitivity of our process. That is, non-RCTs would be better identified as such. A regular expression (regex) parser is used to identify these articles for which the list of PMIDs is saved into a database.

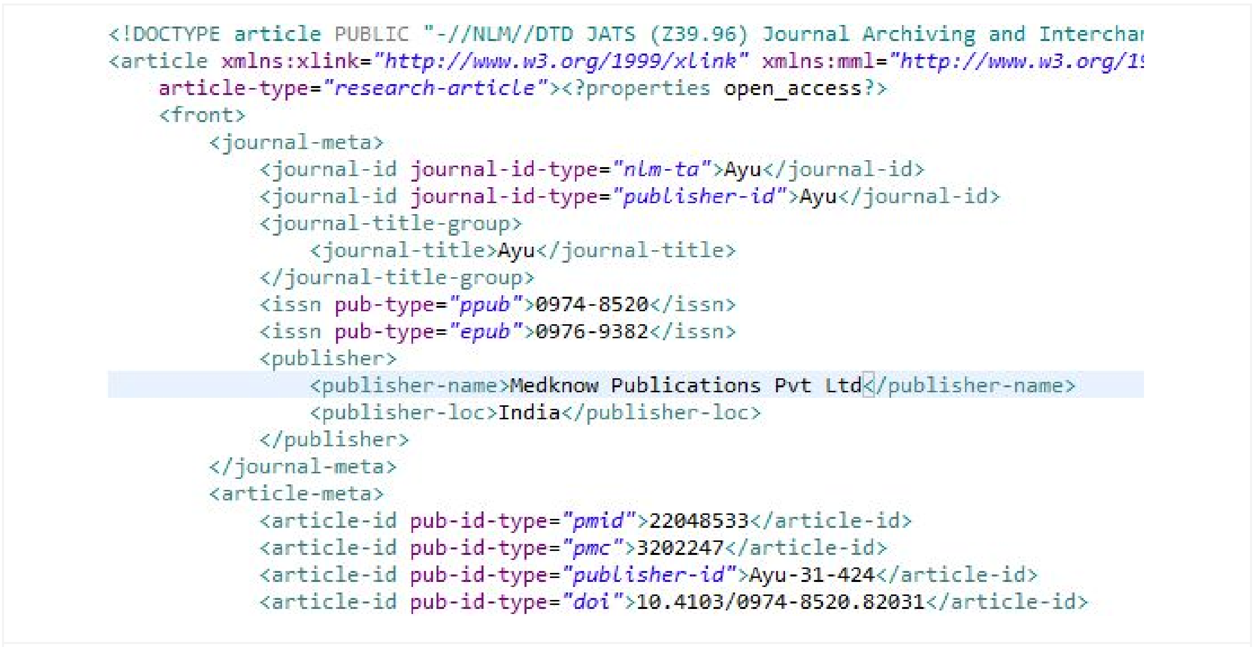
## Matching RCTs to the articles in the PMC Open Access subset

The list of PMIDs for the queried RCTs must match the PMCIDs of the articles in the Open Access (OA) Subset because only the full-text OA articles are used. This OA Subset is part of the total collection of articles in PMC but it allows more liberal redistribution and reuse than a traditional copyrighted work and the full-text of the articles is made available under a Creative Commons or similar license.

Once the PMIDs were matched to their OA PMCIDs, the resulting subset will be the target corpus of the demographic information mining effort. Only files that are in XML format were considered; while the articles are also available in text files, these were not considered due to a reliance on XML tagging. To construct our baseline database, this process reduces the 3.67 million search-returned articles to yield a subset of 301,659 articles in the target corpus.

Article Formats

The XML files have a hybrid structure composed of XML for the text content and HTML for the table content.  In both PubMed and Medline, the XML elements and their attributes have been defined in a Document Type Definition (DTD), which establishes a standard for the document structure and allowable XML elements/tags. The latest version is  pubmed_180101.dtd (as at 01/01/18). Figure 2 shows an example of the top elements from the PMC article with PMC ID3202247:



```
<!DOCTYPE article PUBLIC "-//NLM//DTD JATS (Z39.96) Journal Archiving and Interchai
<article xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:mml="http://www.w3.org/1!
    article-type="research-article"><?properties open_access?>
    <front>
        <journal-meta>
            <journal-id journal-id-type="nlm-ta">Ayu</journal-id>
            <journal-id journal-id-type="publisher-id">Ayu</journal-id>
            <journal-title-group>
                <journal-title>Ayu</journal-title>
            </journal-title-group>
            <issn pub-type="ppub">0974-8520</issn>
            <issn pub-type="epub">0976-9382</issn>
            <publisher>
                <publisher-name>Medknow Publications Pvt Ltd</publisher-name>
                <publisher-loc>India</publisher-loc>
            </publisher>
        </journal-meta>
        <article-meta>
            <article-id pub-id-type="pmid">22048533</article-id>
            <article-id pub-id-type="pmc">3202247</article-id>
            <article-id pub-id-type="publisher-id">Ayu-31-424</article-id>
            <article-id pub-id-type="doi">10.4103/0974-8520.82031</article-id>
```

*Figure 2. PubMed Central sample XML file.*

## Document Mining Pipeline

The text mining and document search components comprise PubMiner's document mining pipeline. To understand how PubMiner works, it is useful to understand how data moves through the system. PubMiner runs a scheduled process to identify documents for demographics mining by using the NCBI e-utilities APIs. Once the documents of interest have been identified, they are downloaded and fed into the document-mining processes which pulls out sentences and tables containing demographic information. This demographic data is persisted to a data store, which the PubMiner web application leverages to enrich standard user searches of PMC/PubMed with demographic data. The data flow diagram in figure 8, below, illustrates this process.

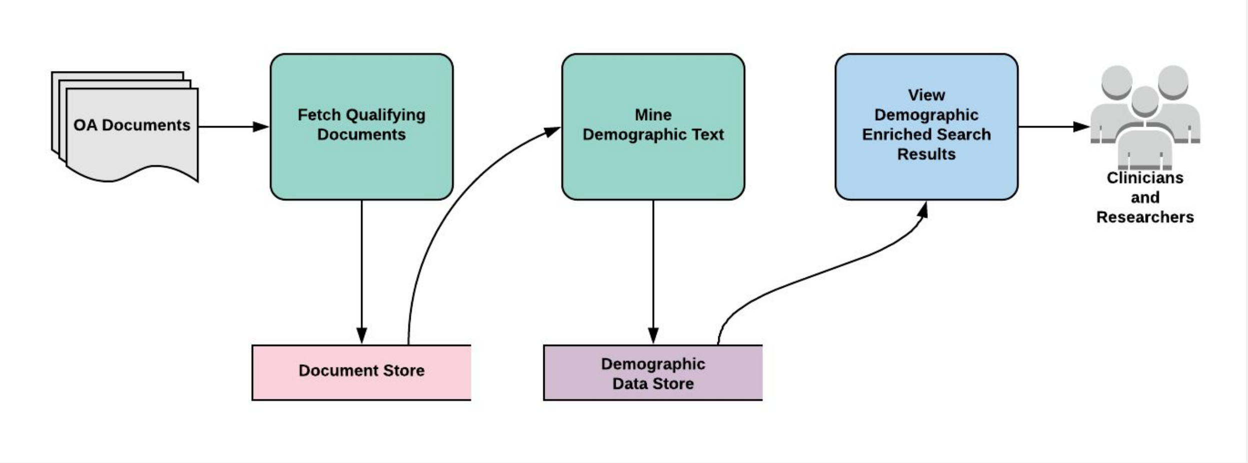
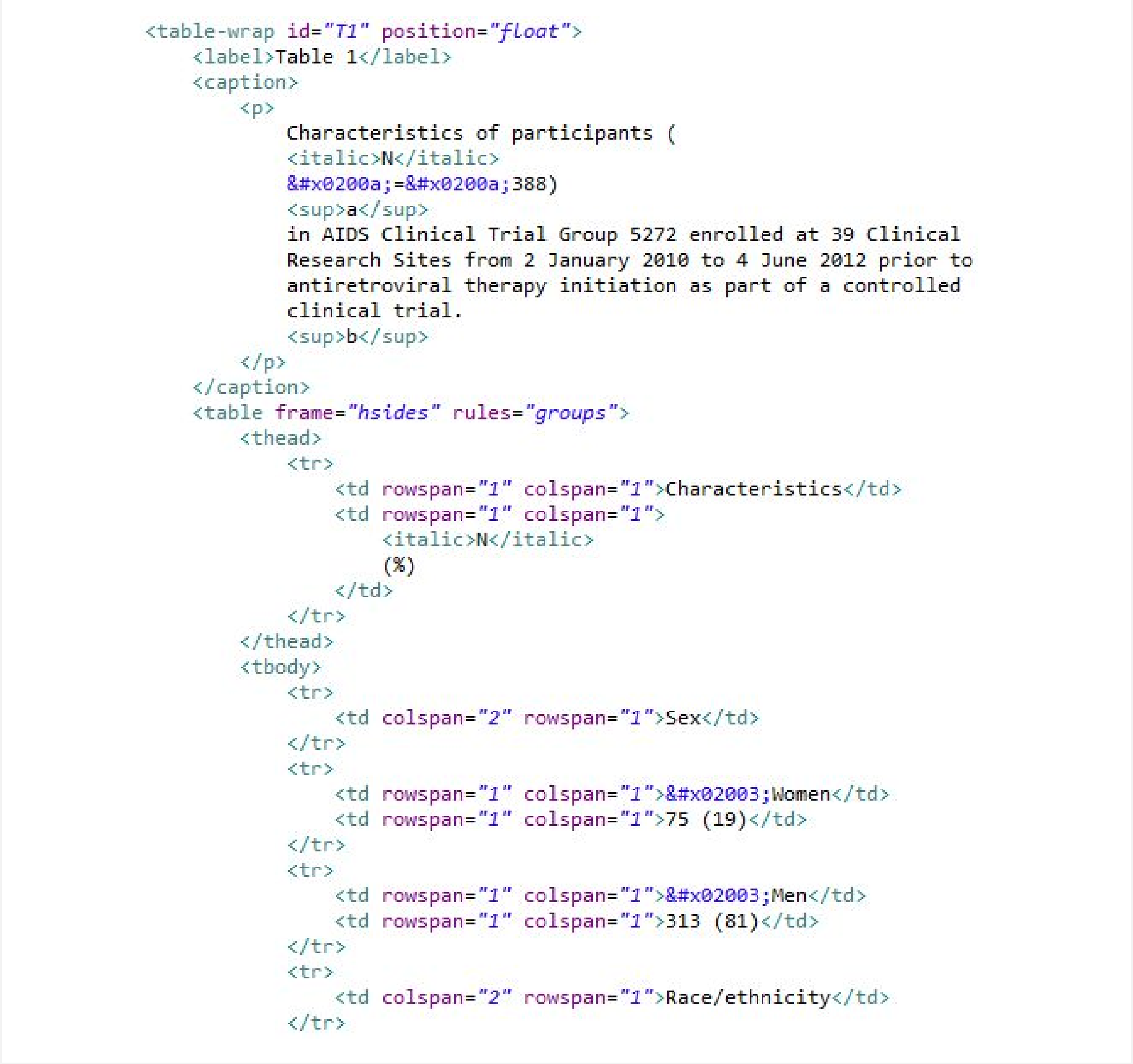

OA Documents
Fetch Qualifying Documents
Mine Demographic Text
View Demographic Enriched Search Results
Clinicians and Researchers
Document Store
Demographic Data Store

*Figure 8. Document mining pipeline*

## Table Mining

Table Structure

For RCTs, demographic information pertaining to trial participants and the randomized groups is embedded in HTML tables within the XML encoding of the article. An example of this is shown in figure 3. This particular example illustrates where demographic information is found within the structured HTML elements of the table itself, but also, in this specific case, in the <caption> of the table which includes the number of participants. While the descriptive measures of trial demographics are often found within the first table of the article, most often called “Table 1”, there is no standard, or even common structure for these tables.



```
<table-wrap id="T1" position="float">
    <label>Table 1</label>
    <caption>
        <p>
            Characteristics of participants (
            <italic>N</italic>
            &#x0200a;=&#x0200a;388)
            <sup>a</sup>
            in AIDS Clinical Trial Group 5272 enrolled at 39 Clinical
            Research Sites from 2 January 2010 to 4 June 2012 prior to
            antiretroviral therapy initiation as part of a controlled
            clinical trial.
            <sup>b</sup>
        </p>
    </caption>
    <table frame="hsides" rules="groups">
        <thead>
            <tr>
                <td rowspan="1" colspan="1">Characteristics</td>
                <td rowspan="1" colspan="1">
                    <italic>N</italic>
                    (%)
                </td>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td colspan="2" rowspan="1">Sex</td>
            </tr>
            <tr>
                <td rowspan="1" colspan="1">&#x02003;Women</td>
                <td rowspan="1" colspan="1">75 (19)</td>
            </tr>
            <tr>
                <td rowspan="1" colspan="1">&#x02003;Men</td>
                <td rowspan="1" colspan="1">313 (81)</td>
            </tr>
            <tr>
                <td colspan="2" rowspan="1">Race/ethnicity</td>
            </tr>
```

*Figure 3. Sample of demographic table information in article XML*

As shown in figure 3, the XML encoding for the article contains a <table-wrap> tag with an id attribute which identifies each table within the document. Generally the id attributes of multiple tables in the document are sequentially ordered, e.g. as "T1", "T2",... However, there is no standard for these id attributes, and it is possible to have such attributes as 'TB1', or some journal-specific identifier, such as "pone-0093540-t001'".

The <table-wrap> element generally has two children: a <label> element preceding a <table> element. The <label> that is of most interest to us would be "Table 1", since this is the table that typically contains demographic information. However, there is no standard labelling convention and this <label> element can also be "Table1", "Tab. 1", "TABLE I", or one of many other variations. The most common is "Table 1", but since selecting on this <label> was not doable, all variations were included.

Beyond these few aspects, there is no standard structure to the tables. Each table adheres to the HTML standard and the structural elements are encoded using HTML tags. However, there is no standard as to the layout/presentation of tables, meaning that header rows can be simple or multi-row, and that the row stubs (e.g. "Mean age (years) (SD)")  can represent single rows or can be super-rows for multi-row variants. The range of tables layouts, from simple lists to complex formats (with for example, spanning super rows and nested cells), makes it very difficult to consistently extract the various relationships from table cells. It is in fact easy to identify cells which contain demographic terms, such as “age”, “adults”, “infants”, “sex”, “gender”, “male”, “female”, “Hispanic”, “Caucasian”, “African-American”, etc. In contrast, it is difficult to determine the roles of cells that are siblings or descendants of such terms. This renders the process of extraction of the numerical information attached to these terms  (mean age in years, standard deviation, percentage share, etc.) a very complex problem.

### Table Truncation

Following the four-step table mining process elucidated by Hurst [Hurst 2000], a focus was placed on only on the first two steps: *table detection* and *functional analysis*, with the third, *structural analysis*, being limited to determining the point at which to truncate the table.  For detection, the <table-wrap> XML tag was relied upon, and including any variation of "Table 1" as a <label> element. For this set of tables, standard form of functional analysis was performed to determine demographic components. Since other tables may also contain demographics, tables with other <label> elements (e.g. "Table 2", "Table S1") were considered; however for these tables a more in-depth form of functional analysis was performed to determine the role of the demographic terms. This is done to avoid inclusion of tables which refer to demographics, for example "age", but do not in fact provide the summary demographic information for the trial. In construction of the baseline database, the table detection brings the subset of 296,400 articles in the target corpus down to 176,200 articles that contain some (any) type of table.

To perform the functional analysis, the lxml library was used to create an element tree (*etree*), and from the etree a search for "age", "sex", "gender", "race" and "ethnicity" using complex regular expression rules was performed. Because "race" and "ethnicity" may not be included as row headings, and a search through the etree for sibling elements in the <table> that contain specific race nomenclatures ("white", "African-American", "Asian", etc.) or common ethnicity groupings ("Hispanic", "Indian") was conducted. This simple form of functional analysis, yields a subset of 27,800 tables that contain "age" and 2,400 tables that contain both "age" and either "sex" or "gender". A slightly more complex rule is used for non-Table 1 tables, where the presence of the demographic terms must be in the row stub and must only appear once.

The last step is the structural analysis, in which tables were truncated using the HTML encoding of the rows. Using lxml, etree was exploited to count the number of rows including demographic terms, and then for each column, only include that number of row siblings, truncating all subsequent siblings. The resulting HTML tree is written to a string in the database.

## Sentence Mining

The overall process of taking in XML files and extracting the sentences that are most likely to contain demographic information is illustrated below. The process begins with taking in XML-formatted articles and converting them into a usable format. For this analysis this means tokenization of sentences. Various features are then extracted from these sentences to create a model that will predict which sentences are most likely to contain demographic information about participants in clinical trials.

Sentence Mining Pipeline

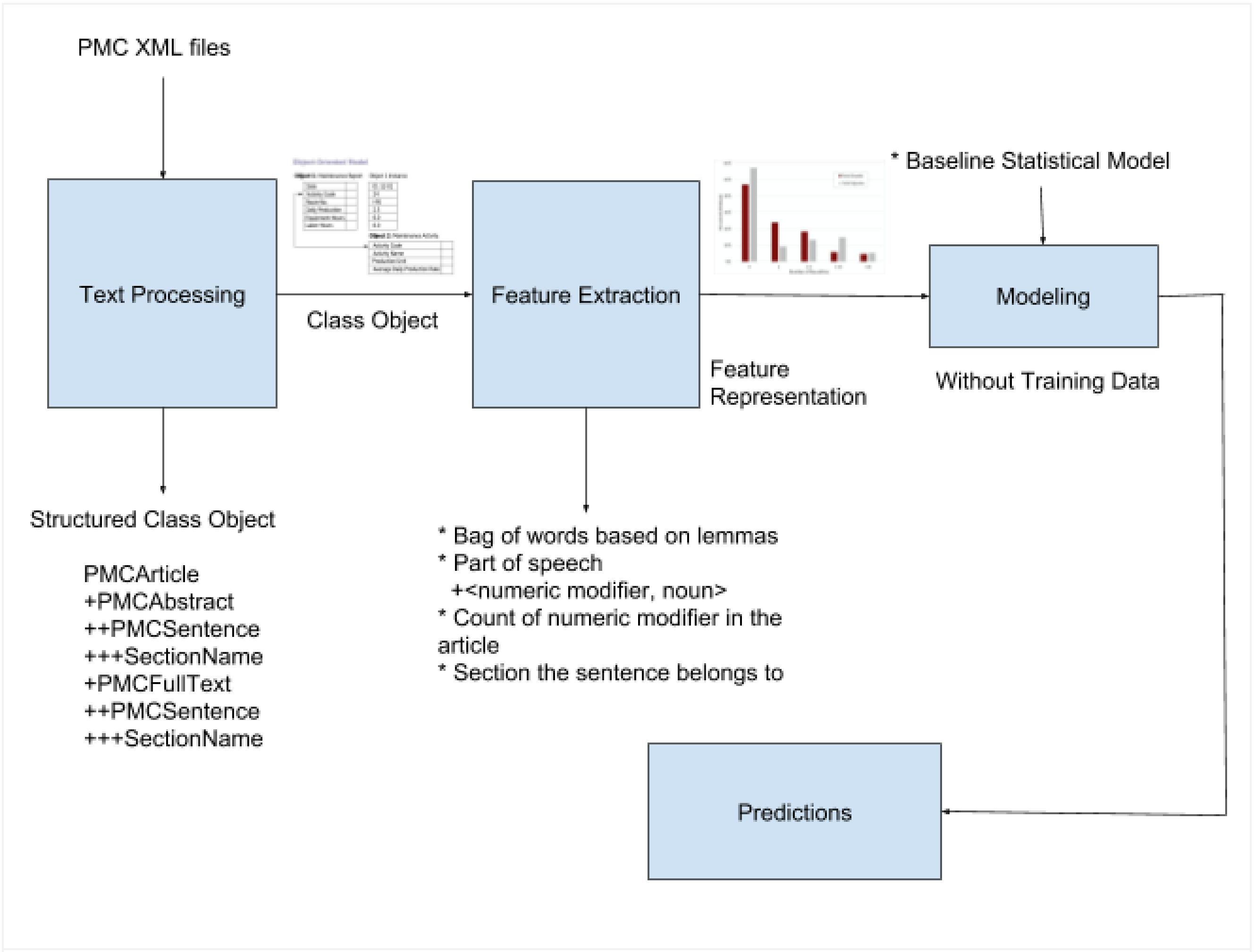


*Figure 4. Sentence mining pipeline*

## Converting Text to Tokenized Sentences

Two tags from these XML files that are of particular interest are the <abstract> and <body> tags. These PMC XML files are of a standard format so an open source PMC parser [Shashank, Agarwal] was used to parse and store these nested XML tags in a Java object.

The Java generalutils' SentenceTokenizer was then used to split the text of the abstract and article text body into a joint collection of sentence strings for future processing. Note that other sentence tokenizer tools could be chosen to break paragraphs into sentences. The generalutils' package SentenceTokenizer worked well with reasonable speeds.

Each word in each sentence was normalized into lemmas, which are groups of the same form of a word (eg., amusement, amusing, and amused), to be used as features of the heuristics and machine learning models. The Stanford NLP toolkits [Manning et. al 2014] were used to obtain lemmas from sentences.

## Initial Sentence Processing

Since analyzing the structure of sentence strings using text mining tools is a computation-heavy job with each article having 200 sentences on average, an initial processing step was conducted using hits of some keywords to do a coarse filtering for demographic sentences. These keywords are "patient", "age", "aged", "male", "female", "subject", "individual", "woman", "man", "women", "men" and "people". "women" and "men" are also included because Stanford NLP toolkit's lemmatization process sometimes doesn't reduce special plural forms such as "women" and "men" into their singular forms, "woman" and "man". This keyword set can likely be expanded for further model tuning.

## Feature Engineering

With sentences prepared in a format ready for further analysis, the focus turns to feature engineering. The following is the list of features used to create the model:

- Part-of-speech tags obtained from Stanford NLP API, especially [numeric modifier, noun] bigrams
- Count of numeric modifier in the article instead of in the sentence
- Bag of words based on lemmas of the sentence
- Sentence's section name

## Statistical Model Feature Weight Tuning

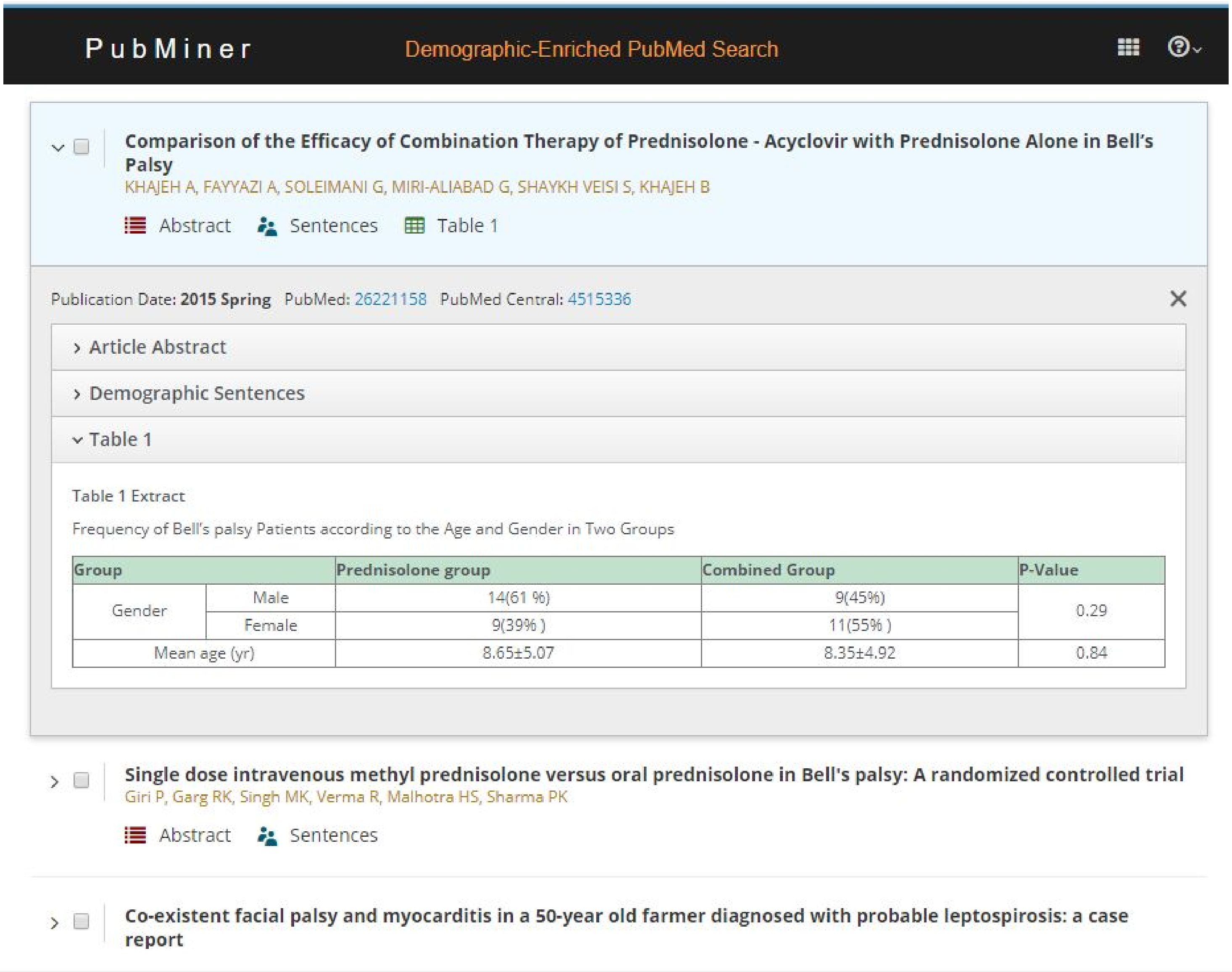
Feature weights are assigned to features identified in the previous section to calculate demographic scores of a sentence by summing them. A higher demographic score means the sentence is more likely to contain demographic information than a sentence having a lower score in the article. These scores are used to select which sentences will be displayed in the web application.

## Results

A web application was made publically available at www.pubminer.org. Figure 5 and 6 show sample results from the web application, including both table and sentence extracts.

At present 301,255 articles were processed for sentence and table extraction. The results of the processing were persisted in a database.

Our sentence mining work was compared with a similar tool(http://www.robotreviewer.net)'s population sentence annotations. Our sentence mining results are consistently better than RobotReviewer's by using RobotReviewer's 3 demo's articles from their home page (26 articles in total). This is the comparison spreadsheet: https://goo.gl/5WrUKo.



**Comparison of the Efficacy of Combination Therapy of Prednisolone - Acyclovir with Prednisolone Alone in Bell's Palsy**
KHAJEH A, FAYYAZI A, SOLEIMANI G, MIRI-ALIABAD G, SHAYKH VEISI S, KHAJEH B

Publication Date: **2015 Spring** PubMed: 26221158 PubMed Central: 4515336

Table 1 Extract

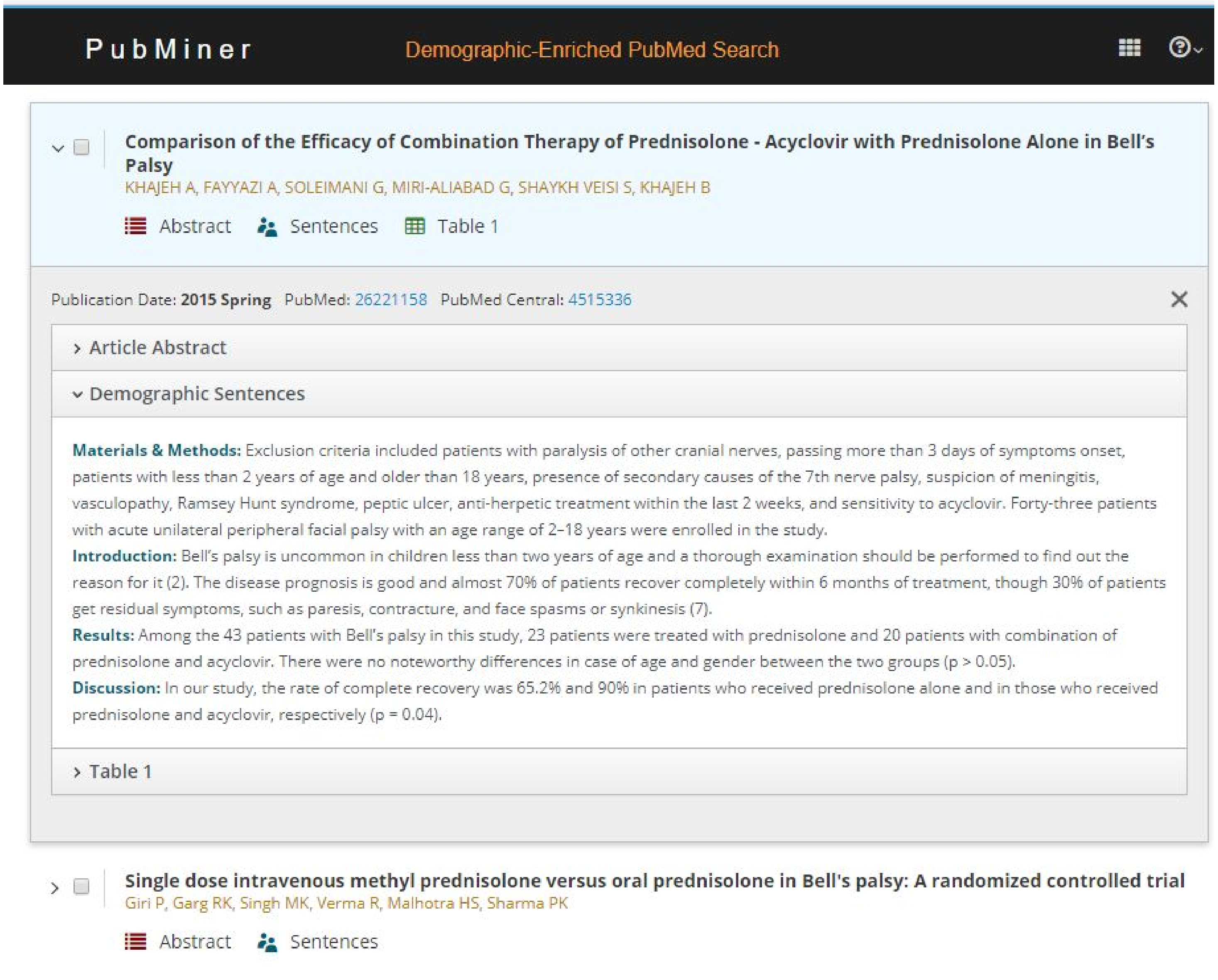
Frequency of Bell's palsy Patients according to the Age and Gender in Two Groups

| Group | | Prednisolone group | Combined Group | P-Value |
|---|---|---|---|---|
| Gender | Male | 14(61 %) | 9(45%) | 0.29 |
| | Female | 9(39% ) | 11(55% ) | |
| Mean age (yr) | | 8.65±5.07 | 8.35±4.92 | 0.84 |

**Single dose intravenous methyl prednisolone versus oral prednisolone in Bell's palsy: A randomized controlled trial**
Giri P, Garg RK, Singh MK, Verma R, Malhotra HS, Sharma PK

**Co-existent facial palsy and myocarditis in a 50-year old farmer diagnosed with probable leptospirosis: a case report**

*Figure 5. UI displaying table 1 extract*



**Comparison of the Efficacy of Combination Therapy of Prednisolone - Acyclovir with Prednisolone Alone in Bell's Palsy**
KHAJEH A, FAYYAZI A, SOLEIMANI G, MIRI-ALIABAD G, SHAYKH VEISI S, KHAJEH B

Publication Date: **2015 Spring** PubMed: 26221158 PubMed Central: 4515336

**Materials & Methods:** Exclusion criteria included patients with paralysis of other cranial nerves, passing more than 3 days of symptoms onset, patients with less than 2 years of age and older than 18 years, presence of secondary causes of the 7th nerve palsy, suspicion of meningitis, vasculopathy, Ramsey Hunt syndrome, peptic ulcer, anti-herpetic treatment within the last 2 weeks, and sensitivity to acyclovir. Forty-three patients with acute unilateral peripheral facial palsy with an age range of 2–18 years were enrolled in the study.
**Introduction:** Bell's palsy is uncommon in children less than two years of age and a thorough examination should be performed to find out the reason for it (2). The disease prognosis is good and almost 70% of patients recover completely within 6 months of treatment, though 30% of patients get residual symptoms, such as paresis, contracture, and face spasms or synkinesis (7).
**Results:** Among the 43 patients with Bell's palsy in this study, 23 patients were treated with prednisolone and 20 patients with combination of prednisolone and acyclovir. There were no noteworthy differences in case of age and gender between the two groups (p > 0.05).
**Discussion:** In our study, the rate of complete recovery was 65.2% and 90% in patients who received prednisolone alone and in those who received prednisolone and acyclovir, respectively (p = 0.04).

**Single dose intravenous methyl prednisolone versus oral prednisolone in Bell's palsy: A randomized controlled trial**
Giri P, Garg RK, Singh MK, Verma R, Malhotra HS, Sharma PK

*Figure 6. UI displaying demographic sentence*

## Future Work

First, a key improvement to the overall system would be to make enhancements to the algorithm for finding the most relevant sentences, whether that be collecting additional data for use in training or further tuning of the model. The current system can be improved both in terms of accuracy in the results and, potentially, the ranking of the results.

An additional possible step would involve implementing a “verification mode” in the web application. The idea here is that users could provide feedback on sentences and articles and whether they are correctly identified and labeled. In essence, this would be crowdsourcing a training data set. This would involve users submitting feedback on suggested sentences to indicate whether they actually contain relevant demographic information. Feedback would then be fed into the system for further model refinement.

From related work on the classification of articles into RCTs and non-RCTs, next steps would include integrating the results of this RCT classification into the PubMiner application as an additional icon, along with that for sentences and tables. The icon would give the percentage probability of the article being a RCT. In addition, a threshold probability (e.g. 50%) could be used to cull suspected non-RCTs from the database, so that users are not given results for articles that are not RCTs.

## References

1. Ardehaly, E.M., Culotta, A. (2017), Co-training for Demographic Classification Using Deep Learning from Label Proportions. Retrieved from https://arxiv.org/pdf/1709.04108.pdf
2. Dickerson, K., Scherer, R., Lefevre, C. (1994), Identifying relevant studies for systematic reviews. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2541778/
3. Entrez Programming Utilities Help [Internet]. Bethesda, MD.: National Center for Biotechnology Information (2010). Retrieved from: https://www.ncbi.nlm.nih.gov/books/NBK25501/
4. Glanville, J. M., Lefebvre, C., Miles, J. N. V., & Camosso-Stefinovic, J. (2006), How to identify randomized controlled trials in MEDLINE: ten years on. Journal of the Medical Library Association, 94(2), 130–136.
5. Govindaraju, V., Zhang, C. (2013), Understanding Tables in Context Using Standard NLP Toolkits. Retrieved from https://cs.stanford.edu/people/chrismre/papers/jointable-acl.pdf
6. Hu, J. Wang, Y., (2002), A Machine Learning Based Approach for Table Detection on The Web. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.11.6758&rep=rep1&type=pdf
7. Hurst, M. F. (2000), The interpretation of tables in texts, PhD thesis. Retrieved from https://www.era.lib.ed.ac.uk/bitstream/handle/1842/7309/515564.pdf
8. Li, Y., McLean, D. (2006), Sentence Similarity Based on Semantic Nets and Corpus Statistics. Retrieved from http://ants.iis.sinica.edu.tw/3BkMJ9lTeWXTSrrvNoKNFDxRm3zFwRR/55/Sentence%20Similarity%20Based%20on%20Semantic%20Nets%20and%20corpus%20statistics.pdf
9. Lu, Z. (2011), PubMed and beyond: a survey of web tools for searching biomedical literature. Database, Vol. 2011, Article ID baq036, doi:10.1093/database/baq036
10. Manning, Surdeanu, Bauer, Finkel, Bethard, McClosky (2014) The Stanford CoreNLP Natural Language Processing Toolkit http://www.anthology.aclweb.org/P/P14/P14-5010.pdf

11. McGibbon, K. A., Wilczynski, N. L., Haynes, B. R., (2009), Retrieving randomized controlled trials from MEDLINE:a comparison of 38 published search filters. Health Information and Libraries Journal, Vol.26, Issue 3, 187-202

12. Milosevic, N. (2016), Disentangling the Structure of Tables in Scientific Literature. Retrieved from https://link.springer.com/chapter/10.1007/978-3-319-41754-7_14

13. Milosevic, N. (2017), Table Mining and Data Curation from Biomedical Literature. Retrieved from https://www.research.manchester.ac.uk/portal/files/32297167/FULL_TEXT.PDF

14. Nishida, K., Sadamitsu, K., Higashinaka, R., Matsuo, Y, (2017), Understanding the Semantic Structures of Tables with a Hybrid Deep Neural Network Architecture. Retrieved from https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/download/14396/13758

15. Pinto, D., McCallum, A. (2003), Table Extraction Using Conditional Random Fields. Retrieved from https://people.cs.umass.edu/~mccallum/papers/crftable-sigir2003.pdf

16. Rink, B., Harabagiu, S. (2011), Automatic extraction of relations between medical concepts in clinical texts. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3168312/

17. Royle, P., Milne, R., (2003) Literature searching for randomized controlled trials used in Cochrane reviews: rapid versus exhaustive searches. International Journal of Technology Assessment in Health Care, 19(4), 591-603.

18. Royle, P., Waugh, N., (2005) A simplified search strategy for identifying randomised controlled trials for systematic reviews of health care interventions: a comparison with more exhaustive strategies. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/16042789

19. Sale, A. M., Chawan, P. M. (2012), Information Extraction from Web Tables. Retrieved from https://pdfs.semanticscholar.org/3ce1/cea5e5497092ea36e3adc73c2674346cc009.pdf

20. Shashank, Agarwal (retrieved March 2018) PMC Parser https://orbit.nlm.nih.gov/browse-repository/software/other/44-pmc-parser

21. Stadermann, J., Symons, S. (2013), Extracting hierarchical data points and tables from scanned contracts. Retrieved from http://ceur-ws.org/Vol-1038/paper_8.pdf

22. Tan, C., Wei, F. (2017), S-Net: From Answer Extraction to Answer Generation for Machine Reading Comprehension. Retrieved from https://www.arxiv-vanity.com/papers/1706.04815/

23. Tchouaa, B. R., Chard, K. (2016), A Hybrid Human-computer Approach to the Extraction of Scientific Facts from the Literature. Retrieved from https://www.sciencedirect.com/science/article/pii/S1877050916307530

24. The Cochrane highly sensitive search strategies for identifying randomized trials in PubMed. (Retrieved March 2018), from Cochrane Work: http://work.cochrane.org/pubmed

25. Wang, Y., Liu, X., Shi, S. (2017), Deep Neural Solver for Math Word Problems. Retrieved from http://aclweb.org/anthology/D17-1088

26. Xu, R., Garten, Y. (2006), Text Classification and Information Extraction from Abstracts of Randomized Clinical Trials: . Retrieved from https://pdfs.semanticscholar.org/645f/d909c18b1aa154a01a3a03de80d8facc52c3.pdf

27. Zhang, C., Shin, J. (2017), Extracting Databases from Dark Data with DeepDive. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5350112/

# RCT Classifier: Applying Deep Learning on RCT Classification

Joe Bockskopf, jbockskopf@g.harvard.edu
Yuning Chen, yuc995@g.harvard.edu
David Dowey, ddowey@g.harvard.edu
Bin Gao, bingao@g.harvard.edu
Alberto Garza, albertogarza22@gmail.com
Ian Smith, ismith864@g.harvard.edu

*Harvard Extension School, Harvard University, Cambridge, MA 02138*

## Abstract

Randomized clinical trials (RCTs) are imperative to the progress of medical research, with thousands being published on an annual basis. Currently, the number of published RCTs is over one million, and half of those RCTs are found in PubMed, a freely accessible search engine primarily accessing the MEDLINE database of biomedical literature. While these RCTs are freely available to the public, there is currently no straightforward way to identify RCTs amongst the multitude of article types in the database. Identifying RCTs efficiently presents itself as an ongoing challenge for medical researchers.

In this paper, a new deep learning model is trained and selected from several models designed for identifying and classifying RCTs. Unlike other related work which use the abstract portions of the papers only, this paper is based on experiments using both the full text documents and their abstracts. The selected deep learning model combines a long short-term memory (LSTM ) model and one-dimension convolution together to process the vectors generated from Doc2Vec. This model can classify articles with a relatively high accuracy and low computational requirement.

**Keywords:** PubMed, PMC, Deep Learning, Topic Classification, Document Classification, Randomized Clinical Trial Classification

[**Submission**: primary journal - Bioinformatics, alternate - BioData Mining]

## Introduction

### Background

Randomized clinical trials are medical studies in which participants are randomly allocated to two or more groups that will receive different interventions, with one group being the classic “control” group. This is done in order to properly compare the effects of interventions and is of utmost importance in medical research. RCTs contain very helpful information regarding the potential treatments for certain disease for both medical researchers and practitioners. At present, more than one million RCTs have been published and approximately half of them are made freely available in a service called PubMed, a freely accessible search engine primarily accessing the MEDLINE database of biomedical literature. While this literature is available to the public, there is currently no way to identify or filter to RCTs for the interested medical researcher or practitioner.

### Literature Review

Relatively few studies have been conducted on how to classify medical literature into either an RCTs of other type of document. Nonetheless, the following is a discussion of relevant work that has been conducted in the area.

An RCT dataset for sequential sentence classification was created [Dernoncourt, Franck 2017] to indicate the role of each sentences in an RCT abstract.

Machine learning was applied to classify RCTs among other medical documents [Marshal et al. 2017]. The result indicates that using a support vector machine (SVM) and a convolutional neural network (CNN) on abstracts can separate RCTs well from other medical documents.

SVM and Associative Text Classification (ATC) are used to classify clinical trial reports [Sarioglu, Efsun 2013], and the result indicates that SVM performs better than ATC.

From a more general perspective, RCT classification belongs to the work of topic classification. SVM and Naive Bayes are traditionally used in this task. Naive Bayes is well known for its speed and high accuracy for topic classification [Ting, S 2011].

However, Naive Bayes does not always mean the best solution. A comparison between Naive Bayes and SVM [Wang, Sida, 2012] indicates that Naive Bayes can give better result on snippets, while SVM outperform Naive Bayes on full lengh reviews.

For long text classification, linear discriminant analysis (LDA) is used to do dimension reduction and SVM is used to do classification [Li, Kunlun 2011]. Moreover, LDA can also improve the performance in datasets with high sparseness issue in short text topic classification [Chen, Qiuxing 2016].

Deep learning is also used to do paragraph classification. A growing body of recent research indicates that both CNN and LSTM can get a good score for paragraph classification. While a CNN runs significantly faster, an LSTM can give better results [Nho, Eugene 2016].

An effective tool to deal with sequence data is the LSTM model [Sepp Hochreiter, 1997]. This model can be seen as an upgraded recurrent neural network (RNN) model. It contains three gates: drop out, input, and output gate. An LSTM can transfer the information from a much earlier step in a sequence to other LSTM units. Therefore, this ‘memory’ helps to manage the vanishing gradient issue for the deep RNN.

A bottleneck for training LSTMs is present in the downsizing of the sequence input. In order to address this bottleneck, one solution [Ng 2018] is to introduced a 1-D convolutional layer as an initial layer in the network, in order to downsize the vectors that will be inputs to subsequent gated recurrent layers (GRU).

To generate the vectors of an article, the traditional approach is N-grams and TF-IDF. The hidden assumption of N-grams is it based on a Markov Model. TFIDF is the combination of the Terms’ frequency [Luhn, Hans Peter, 1957] and inverse document frequency[Spärck Jones 1972] to evaluate the importance of a word for a document in a set of corpus.

There have also been more recent developments in vectorization by way of Word2Vec [Mikolov, Tomas 2013] and Doc2Vec [Mikolov, Tomas 2014], which are predictive models for learning word embeddings from raw text.

Word2Vec uses skip-gram or a CBOW (continuous bag-of-words) to generate the word vectors, in order to capture the relationship between different words effectively. However, this algorithm has two limitations: first, the word order information is lost, and second, the word semantics cannot be kept either. In order to solve these limitation, Doc2Vec was developed in 2014, and can generate vectors containing more semantics and sequence information.

## Methodology

The entire process of developing an RCT classification tool can be separated into four distinct steps: fetching data, tokenization of sentences, vectorization of sentences, and finally classification. These parts are illustrated in the figure below:

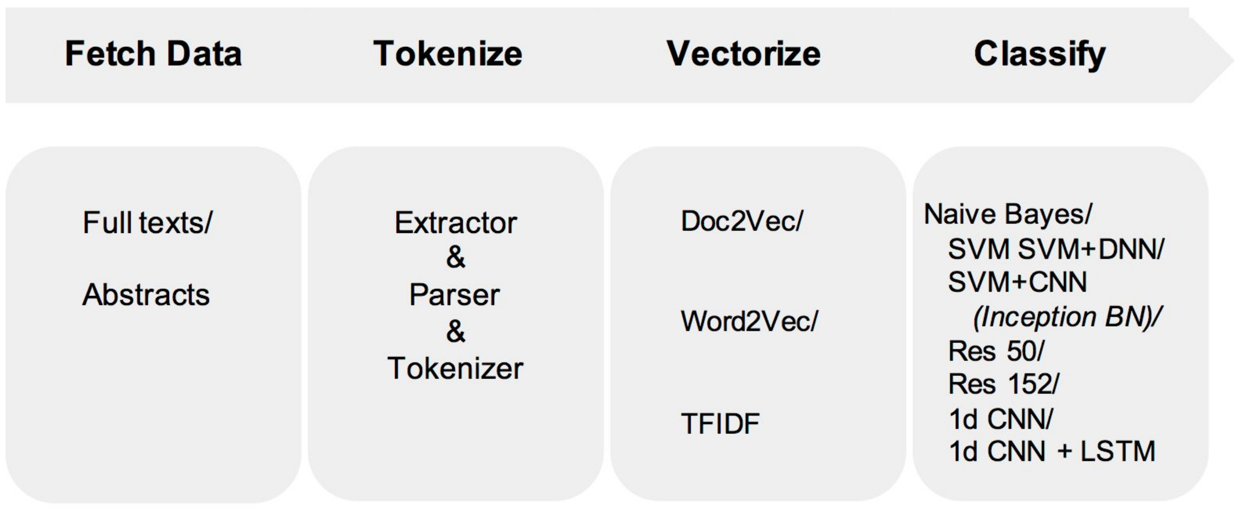


*Figure 7. Classification is separated into four distinct steps*

Step 1:

First a dataset composed of both labeled RCTs and non-RCTs is prepared for further processing. After combining the 171k labeled RCT/non-RCT labels from clinicaltrials.gov and the 1.7m open access files on the NCBI website, approximately 10k articles are identified as candidates for use in the dataset. Approximately 2,000 are non-RCTs and 8,000 are RCTs, making an unbalanced dataset. Although training a model using unbalanced sampling results in an F1 score of about 93%, further verification using real world data is desirable.

Borrowing from related work that identified both tables and sentences from Pubmed articles [Pubminer 2018], additional articles with high likelihood of being non-RCTs were then added to produce a more balanced dataset containing 19,111 articles. These articles were split into training and testing sets for the training models.

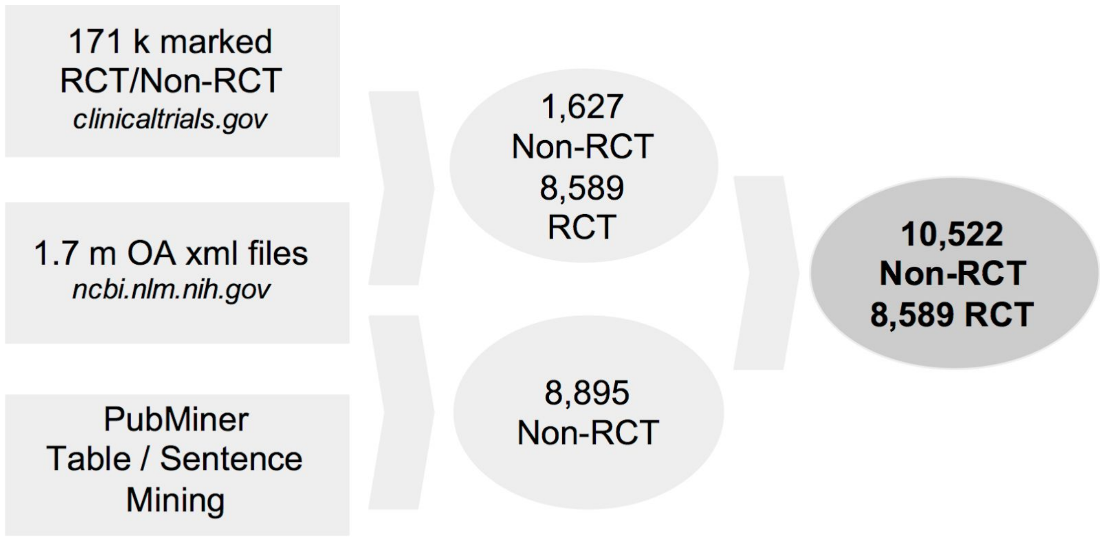


*Figure 8. Summary of the construction of a dataset to be used to train models*

Step 2:

Having a dataset or articles identified, a Python package, BeautifulSoup, was used to scrape the abstracts and full text from the set of 19,111 articles. Pre-processing to remove symbols and stop words was conducted prior to constructing the corpus on which models would be trained.

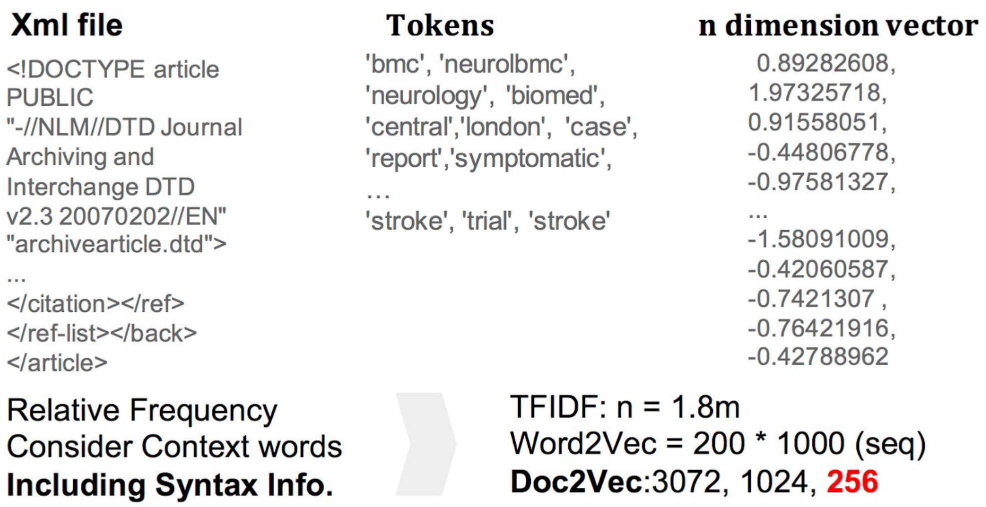


*Figure 9. Text processing was needed before any modelling could be conducted.*

Step 3:

With a corpus prepared, vectors were prepared using three vectorization tools: Doc2Vec, Word2Vec, and TFIDF. For Doc2Vec, generation of vectors was conducted using three different parameters for the sizes of the full text dataset: 256, 1024, and 3072. For Word2Vec, a pretrained RCT sentence vectors [Dernoncourt, Franck, 2017] was used. Among the models using vectorization, the best performance in terms of accuracy and F1 score is the Doc2Vec model.

The fastest Doc2Vec tool comes from the gensim package, which uses fast mapping technology and parallel computation to speed the calculation to be even faster than other implementations that use graphics processing units (GPUs) for processing. However, it is still time and memory consuming to use the Doc2Vec algorithm for big vectors. Generating the Doc2Vec vectors with a size of 3072 takes about 12 hours occupying all the threads of two workstation-grade XEON processors and approximately 20GB of memory. Generating the Doc2Vec vectors with a size of 256 only takes about 1 hour and 10GB of memory using the same hardware.

Different windows, epochs, and min_counts were also attempted for constructing the Doc2Vec vectors. During the experiments, it was found that even a vector size of 256 can generate the favorable results as long as other parameters were set appropriately. Setting a min_window of 1 can help the Doc2Vec vectors capture information more sensitively while increasing the window of context length to 21 helps to preserve more syntax information. The epochs were set to 50 in the final solution.

The Python sklearn package was used to test the term frequency–inverse document frequency (TF-IDF) for its convenience and efficient use as a pipeline solution. This combination also yields the best results when only using the abstract portions of the articles. The results validate the findings of Wang [Wang, Sida, 2012] that Naive Bayes can yield better results on snippets.

A pre-trained RCT using Word2Vec vectors was also used. The vector size was wet to 200 and sequence sizes of 100 and 1000 were tested. Keras, a high-level neural network tool, was used to implement the LSTM model. This model did not outperform the model using Doc2Vec vectors in the experiments. One potential explanation may be that this vector was trained using sentences from the abstract portions of the articles only. Another possible reason is that the dataset is too small and there may be some overfitting of the model.

However, after trying other simpler LSTM structures for the dataset similar results were obtained. In the end, the pre-trained RCT Word2Vec was not used. The hypothesis here is that losing the sequence and semantics information might be the root cause of the poorer performance.

Using an initial embedding layer substantially increases the size of the LSTM, requiring significantly longer training times for the network model. On one NVIDIA GTX1060 6G GPU it took approximately 4 hours to train the LSTM model to produce a final F1-score of 83% on the test set.

Step 4:

After extensive testing of different options, the final model selected was a Naives Bayes classifier, SVM and nine different deep learning networks using word or document vectors. Below are a few samples of the model structures. Full figures can be found in the Appendix.

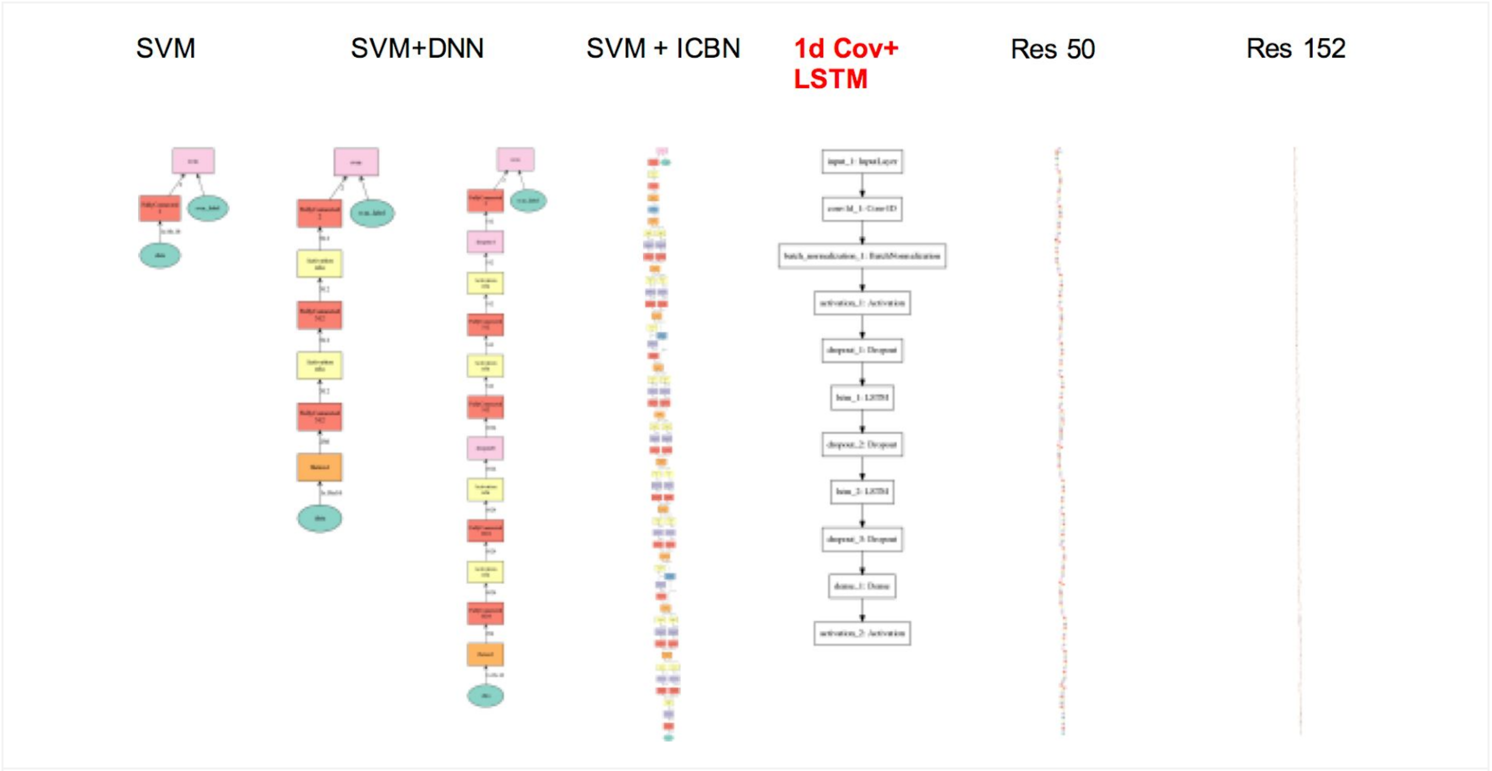


*Figure 10.*

MXNET was the deep learning API package chosen to implement all the deep learning models except the LSTM, for which Keras was used. The reason is that MXNET is faster than other deep learning packages on a GPU machine, and its symbolic language is user-friendly. Models that were used in these experiments were SVM, along with two different deep neural networks combined with SVM, as well as the Inception CNN model with Batch Normalization. Both DNN gives the similar result (F1 Score 84% to 85%). The simplest deep neural networks can be trained relatively fast, about 0.25 seconds per epoch (for 256 vector size), and these were trained for 300 epochs. The bigger DNN model is not slow either, about 0.38 seconds per epoch (for 256 vector size). However, the Inception CNN model with Batch Normalization did not provide superior performance, despite taking about 4.5 seconds per epochs to produce a 83% F1 score.

Transfer learning was also attempted to use a pre-trained ResNet-50 and ResNet-152 model. These models result in no more than an 80% F1 score. One reason for this might be that the dataset is too small in terms of both vector size and article count for the Inception Batch Normalization and Res-Net models. Another reason might be that they are networks with 2-D convolutional layers which may not suit language processing and that a network architecture with a 1-D convolutional layer might fit better.

The dimensions of the vectors generated by TF-IDF and Word2Vec can be very large. In testing, these were 1.8M and 200* 1000 sequences, respectively.  Therefore, it was key to investigate how to downsize the vectors in order to to effectively train the LSTM models.

Inspired by the application of 1-D convolutional layers as the initial network layers to downsize the audio signals [Ng 2018], the below LSTM model was designed.

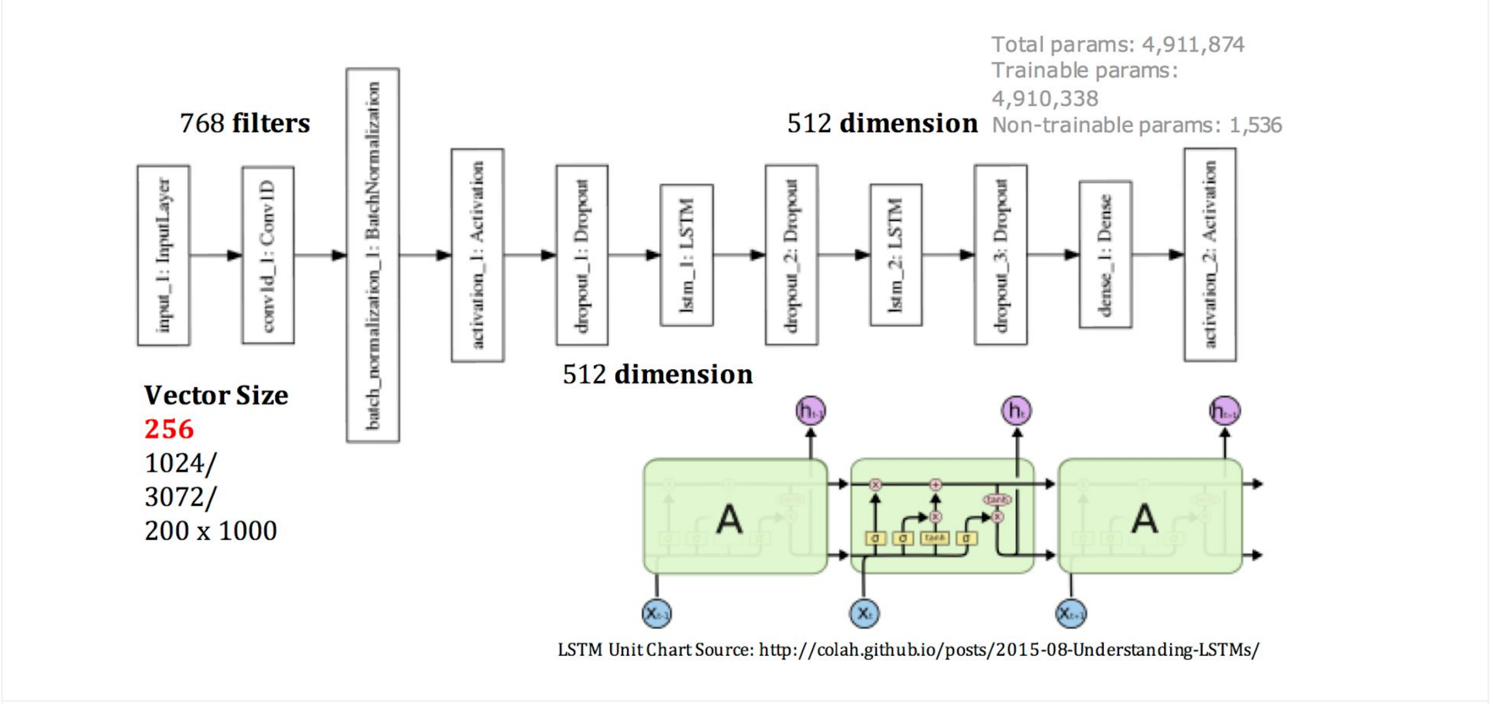


*Figure 11. Illustration of the LSTM network design*

This model contains a 1-D convolutional layer, two LSTM layers, and the corresponding dropout, activation, and normalization layers. Benefitted from the 1-D convolutional layer, the LSTM units can be reduced remarkably. For instance, if using a 16x16 vector to the LSTM without the 1-D convolutional layer, the first layer of the LSTM contains 16 units with 512 dimension. After using the 1-D convolutional layer there was only 1 LSTM in both LSTM layers if filter size of the 1-D convolutional layer is set to 15.

## Results

The full details of the various experiments can be found in the attachment.

For the abstracts data, a Naive Bayes model was the best solution. It yields a F1-score of 81%. For full text data, a one-dimension convolution + LSTM is the winning solution, which yields 86% for precision, recall, and F1-Score.

Doc2Vec helps to generate good vectors fed to this network. Instead of increasing the vector size, adjusting other hyperparameters such as window, min_count, and epoch can yield good results with fewer time a memory resource consumption.

The 1-D convolutional layer is important with the long word sequences used. This was in order to reduce the dimensions of the vectors to use in an LSTM model and therefore to maintain performance while reducing both training time and resource needs.

## Future Work

Due to a limited amount of time, our experiments do not cover the full range of combinations of both the deep network architectures and the methods to vectorize the text content.

A new development of language process comes from deep learning sequential models with an attention component [Dzmitry Bahdanau, 2014]. Attention models were not investigated, however the impressive results of these models on text analysis problems, including document classification, lead us to consider this as the primary avenue for future work.

Lastly, since the simple Naive Bayes classifiers had relatively superior performance, ensemble methods with Naive Bayes combined with sequential models or other non-Neural Network classifiers would be an promising and interesting topic to investigate further.

## References:

1. Iain J. Marshall, Anna Noel-Storr, Joël Kuiper, James Thomas, Byron C. Wallace, (2017) Machine learning for identifying Randomized Controlled Trials: An evaluation and practitioner's guide. https://onlinelibrary.wiley.com/doi/full/10.1002/jrsm.1287

2. Franck Dernoncourt, Frank, Lee Ji Young (2017), PubMed 200k RCT: a Dataset for Sequential Sentence Classification in Medical Abstracts. Retrieved from https://arxiv.org/pdf/1710.06071.pdf

3. Efsun Sarioglu, Kabir Yadav, Topic Modeling Based Classification of Clinical Reports, http://www.aclweb.org/anthology/P13-3010

4. Ting, S. L., Ip, W. H., Tsang, A. H. (2011). Is Naive Bayes a good classifier for document classification, https://www.researchgate.net/publication/266463703_Is_Naive_Bayes_a_Good_Classifier_for_Document_Classification

5. Sida Wang and Christopher D. Manning, Baselines and Bigrams: Simple, Good Sentiment and Topic Classification, https://www.aclweb.org/anthology/P12-2018

6. Mengen Chen, Xiaoming Jin, Short Text Classification Improved by Learning Multi-Granularity Topics, http://www.ijcai.org/Proceedings/11/Papers/298.pdf

7. Kunlun Li, Jing Xie, Multi-class text categorization based on LDA and SVM, https://www.sciencedirect.com/science/article/pii/S1877705811018674

8. Qiuxing Chen, Lixiu Yao, 2016, Short text classification based on LDA topic model, https://ieeexplore.ieee.org/document/7846525/

9. Eugene Nho, Andrew Ng., Paragraph Topic Classification

10. http://cs229.stanford.edu/proj2016/report/NhoNg-ParagraphTopicClassification-report.pdf

11. Andrew Ng. Sequence Models, https://www.coursera.org/learn/nlp-sequence-models

12. Spärck Jones, K. (1972). "A Statistical Interpretation of Term Specificity and Its Application in Retrieval". Journal of Documentation. 28: 11–21. doi:10.1108/eb026526.

13. Luhn, Hans Peter (1957). "A Statistical Approach to Mechanized Encoding and Searching of Literary Information" (PDF). IBM Journal of research and development. IBM. 1 (4): 315. doi:10.1147/rd.14.0309.

14. Mikolov, Tomas; et al. "Efficient Estimation of Word Representations in Vector Space". https://arxiv.org/abs/1301.3781

15. Quoc Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. http://arxiv.org/pdf/1405.4053v2.pdf

16. Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation. 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. PMID 9377276.

17. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, Neural Machine Translation by Jointly Learning to Align and Translate, https://arxiv.org/abs/1409.0473

## Appendix:

1. Experiments Overview:

**1. Experiment on Abstracts only**

| **Vectorizer** | **Classifier** | **Precision** | **Recall** | **F1-Score** |
|---|---|---|---|---|
| TFIDF | Naive Bayes | 82% | 81% | 81% |
| DOC2VEC (size = 1024 Window = 3 Iter = 30 Min Count =2) | SVM | 77% | 77% | 77% |
| | DNN+SVM | 74% | 74% | 74% |
| | DDNN+SVM | 75% | 75% | 75% |
| | Inception Batch Normalization | 74% | 74% | 74% |
| | 1dCNN + LSTM | 78% | 78% | 78% |

**2. Experiment on Full text without the 9k non-RCT from PUB Miner**

| **Over-Sampling** | **Vectorizer** | **Classifier** | **Precision** | **Recall** | **F1-Score** |
|---|---|---|---|---|---|
| No Over Sampling | DOC2VEC (size = 1024 Window = 3 Iter = 30 Min Count =2) | 1dCNN + LSTM | 79% | 84% | 78% |
| SMOTE | | SVM | 80% | 80% | 80% |
| | | DNN+SVM | 93% | 93% | 93% |
| | | DDNN+SVM | 93% | 93% | 93% |
| | | Inception Batch Normalization | 94% | 94% | 94% |
| | | 1dCNN + LSTM | 93% | 93% | 93% |

**3. All the follow experiments include the 9k non-RCT from PUB Miner**

| Over-Sampling | Vectorizer | Classifier | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| No Over Sampling | DOC2VEC (size = 1024 Window = 3 Iter = 30 Min Count =2) | SVM | 81% | 81% | 81% |
| | | DNN+SVM | 84% | 84% | 84% |
| | | DDNN+SVM | 83% | 83% | 83% |
| | | Inception Batch Normalization | 81% | 81% | 81% |
| | | 1dCNN + LSTM | 86% | 85% | 85% |

**4. Modify Doc2vec hyper parameters**

| Over-Sampling | Vectorizer | Classifier | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| No Over Sampling | DOC2VEC (size = 1024 Window = 8 Iter = 50 Min Count =1) | SVM | 85% | 83% | 83% |
| | | DNN+SVM | 85% | 85% | 85% |
| | | DDNN+SVM | 84% | 84% | 84% |
| | | Inception Batch Normalization | 81% | 81% | 81% |
| | | 1dCNN + LSTM | 85% | 85% | 85% |

**5. Increase Doc2vec vector size**

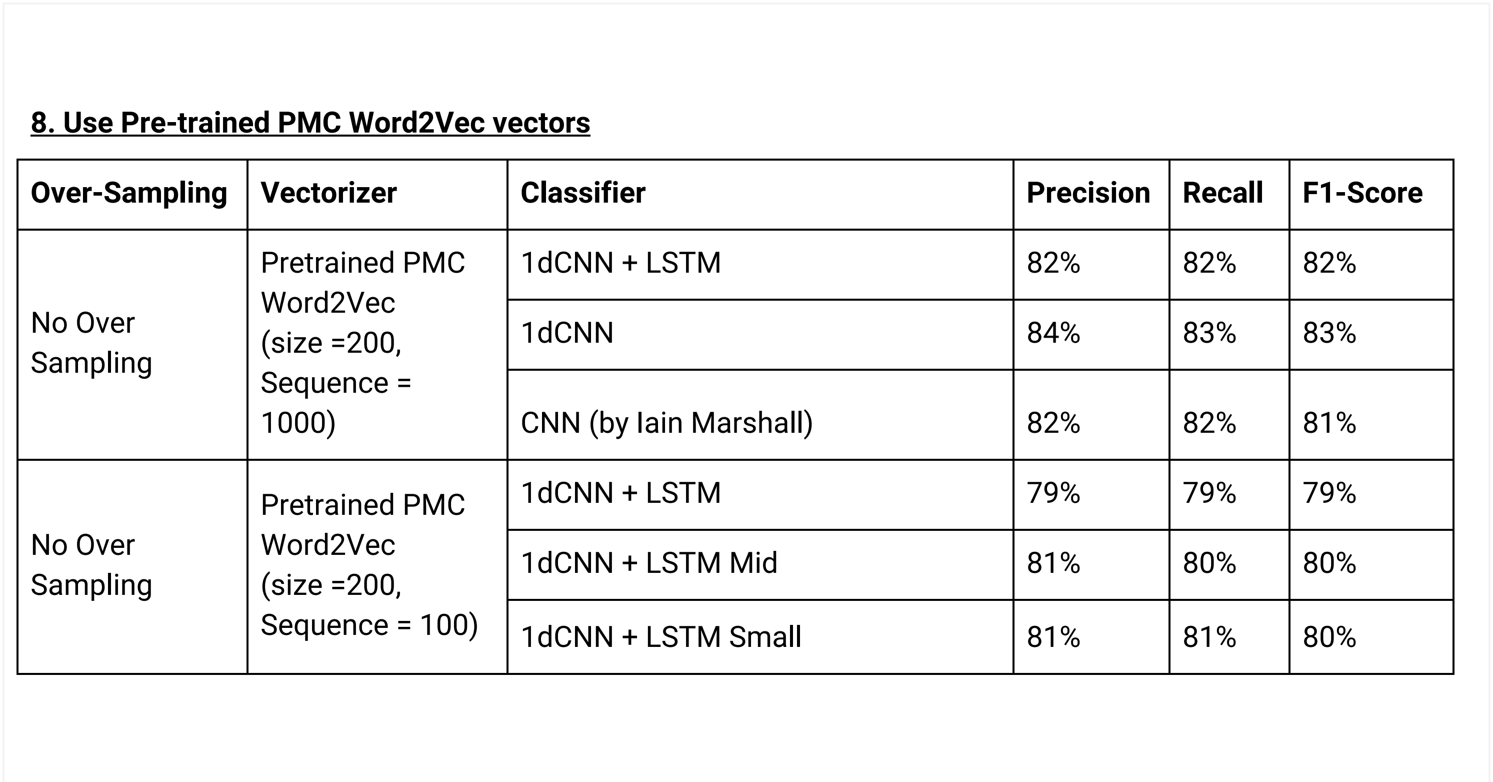
| Over-Sampling | Vectorizer | Classifier | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| No Over Sampling | DOC2VEC (size = 3072 Window = 21 Iter = 50 Min Count =2) | SVM | 85% | 85% | 85% |
| | | DNN+SVM | 85% | 85% | 85% |
| | | DDNN+SVM | 85% | 85% | 85% |
| | | Inception Batch Normalization | 81% | 81% | 81% |
| | | 1dCNN + LSTM | 86% | 85% | 86% |

**6. Implement Transfer Learning**

| Over-Sampling | Vectorizer | Classifier | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| No Over Sampling | DOC2VEC (size = 3072, Window = 21, Iter = 50 Min Count =2) | Res 50 | 75% | 75% | 75% |
| | | Res 152 | 75% | 74% | 75% |

**7. Decrease Doc2vec vector size**

| Over-Sampling | Vectorizer | Classifier | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| No Over Sampling | DOC2VEC (size = 256 Window = 21 Iter = 50 Min Count =1) | SVM | 84% | 84% | 84% |
| | | DNN+SVM | 85% | 84% | 84% |
| | | DDNN+SVM | 85% | 85% | 85% |
| | | Inception Batch Normalization | 83% | 83% | 83% |
| | | 1dCNN + LSTM | 86% | 86% | 86% |



**8. Use Pre-trained PMC Word2Vec vectors**

| Over-Sampling | Vectorizer | Classifier | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| No Over Sampling | Pretrained PMC Word2Vec (size =200, Sequence = 1000) | 1dCNN + LSTM | 82% | 82% | 82% |
| | | 1dCNN | 84% | 83% | 83% |
| | | CNN (by Iain Marshall) | 82% | 82% | 81% |
| No Over Sampling | Pretrained PMC Word2Vec (size =200, Sequence = 100) | 1dCNN + LSTM | 79% | 79% | 79% |
| | | 1dCNN + LSTM Mid | 81% | 80% | 80% |
| | | 1dCNN + LSTM Small | 81% | 81% | 80% |

*Figure 12. Experiment details*

## 1. System Design

### Overview

The PubMiner system consists of three logical components: one for identifying and downloading publications of interest, another that performs text mining and demographic data extraction on the downloaded publications, and, finally, a web service and user interface that deliver enriched PubMed queries by augmenting PubMed search results with the extracted demographic data stored by the data extraction process.

These components are highlighted in figure 13, which provides a high-level view of how the PubMiner system relates to NCBI and PubMed Central.

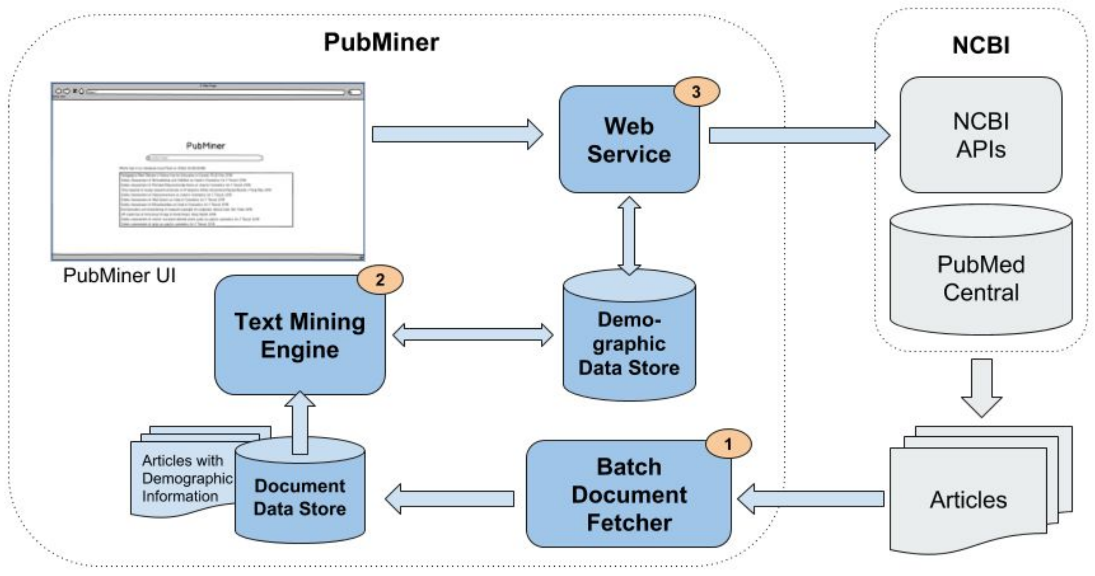


*Figure 13. Relationship of PubMiner system to NCBI*

Document-Mining Architecture

The document-mining architecture is implemented as loosely-coupled tasks that are tied into a workflow. Several AWS services are used to orchestrate and execute the workflow.

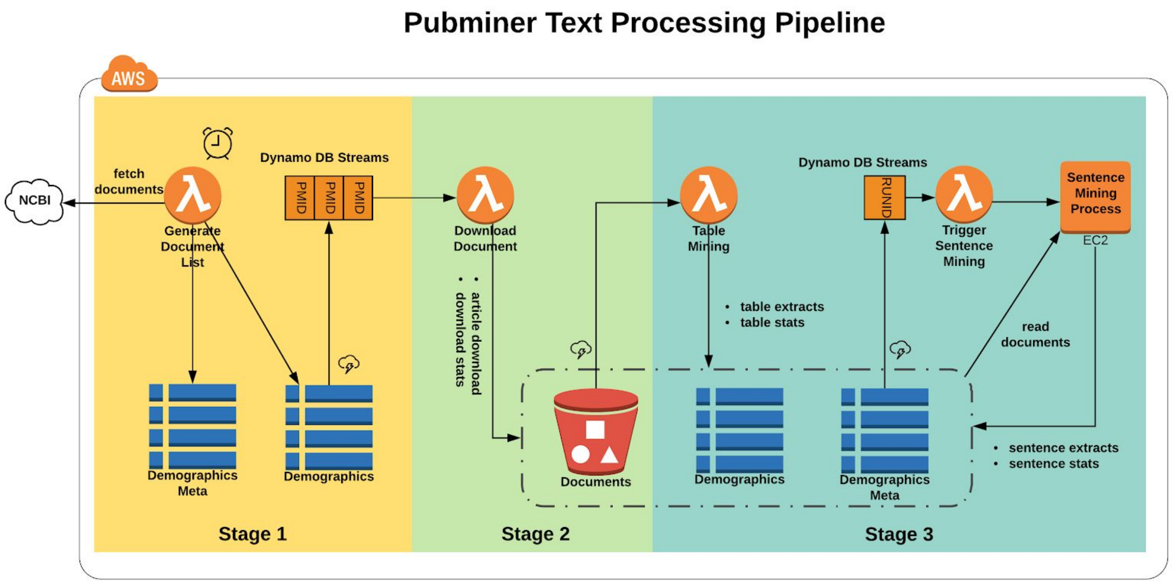


*Figure 14. Document mining architecture*

Figure 14 illustrates the system architecture used for the document-mining pipeline. Reading from left to right, the pipeline maps closely to the data flow depicted in figure 13. AWS was chosen for these tasks given the rich set of platform tools that it provides as well as the ability to create workflows from disparate components. By using services such as AWS Lambda, we are able to focus on the core algorithmic aspects of extraction, getting the benefits of the platform's capability and scalability out of the box.

The detailed steps for running the document workflow are as follows:

1. A scheduled CloudWatch event (every 24h) occurring at 6:00 GMT triggers an AWS Lambda function (GetPMCUpdatesFromCSV) to run Python code to process any new articles that satisfy the search using the RCT query string and that are available in PMC OA Subset. Only articles that have a Entrez publication date within the last 2 days are considered as new, which allows some overlap between daily searches. The first step in the process is to issue the pre-defined search query (see figure 1) to the eutils esearch API to get a list of PMIDs.

2. All articles available in the Open Access subset, including the newest updates, are contained in a .csv file available on the NCBI FTP site. This file is downloaded and the list of PMIDs is intersected with the Open Access subset to get a list of the PMIDs and their corresponding PMCIDs. This list represents the newly published articles that should be processed.

3. The current date, the PMIDs and the PMCIDs (primary key) are written as an item to the dynamodb table “*demographics*”. The metadata item for demographics in the “*demographics_meta*” table is updated with the information from the search, and counters are reset to zero for downloads, number of tables extracted, and the number of sentences extracted. Both of these dynamodb tables have been configured with streams to trigger subsequent AWS Lambda functions. These first three steps are part of the same Lambda function and comprise Stage 1 of the process.

4. Next, the AWS Lambda function for Document Download (DownloadPMC-OAFileToS3) polls the dynamodb stream from the *demographics* dynamodb table. The INSERT or MODIFY items in the stream will trigger the function, which, one by one, downloads the target articles from the PMC-OA website, uncompresses the file from tar.gz format and then uploads the .nxml file to the S3 bucket “*pubmedcentral_oa*”. The function also updates the statistics, by incrementing the counter *items_downloaded* in the *demographics_meta* dynamodb table. There is also a trigger action for the next Table Mining step. This step comprises Stage 2.

5. The AWS Lambda function for the table truncation (TruncateTable) is triggered for each article .nxml file put to the S3 bucket “*pubmedcentral_oa*”. The function opens the .nxml file from S3, then extracts the relevant demographic information from the table and writes the resulting table as an HTML string to the table1 attribute of the article’s item in the *demographics* dynamodb table. It also increments the with_tables counter in the *demographics_meta* dynamodb table in order to update the statistics.

6. An AWS Lambda function for sentence extraction (SentenceMinerOnEC2instance) polls the stream from the *demographics_meta* dynamodb table, waiting for the event that the number of articles put to S3 (*items_downloaded*) equals the number of new updates (*items_updated*). In this way, the end-of-stream event for the updates to the *demographics* dynamodb table is captured, such that we can be assured that the full set of article .nxml files have been uploaded to S3, necessary for the batch sentence extraction to execute. This threshold event triggers the function to launch a new EC2 instance in order to perform the sentence extraction and write the output to the sentences attribute in the *demographics* dynamodb table. The function also increments the counter *with_sentences* in the *demographics_meta* dynamodb table, to maintain the statistics. When the full set of updates has been processed, the instance signals task completion (by updating a file on S3) and then self-terminates. The steps 5 and 6 comprise Stage 3 of the process.

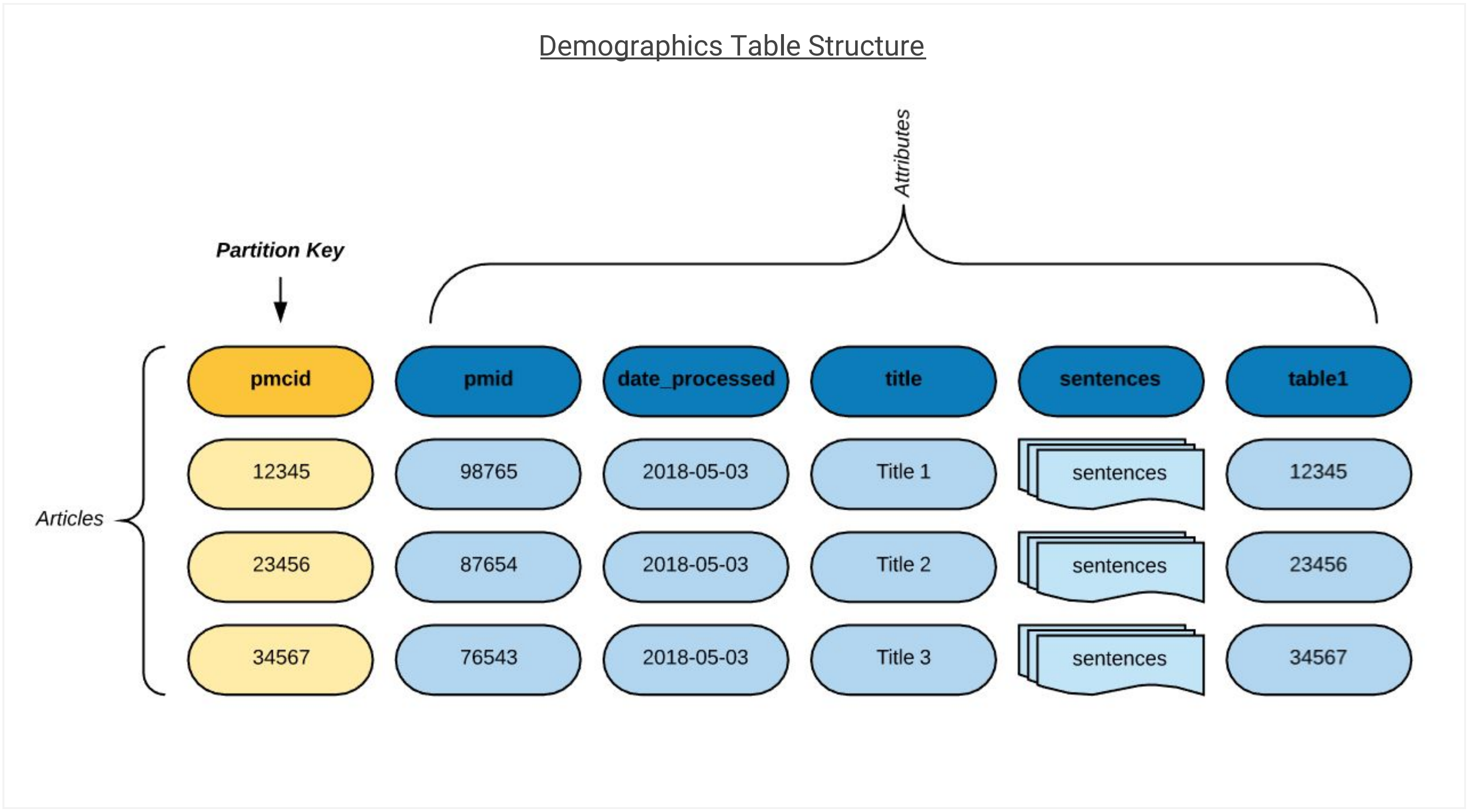
7. At the end of the process, the *demographics* dynamodb table has been updated with all the items for all articles in the update, including the attributes for extracted tables and sentences, where those exist. The *demographics_meta* dynamodb table has been updated to include statistics on the current number of items in the database, the number of updates, and download, and how many articles in the update have sentences and tables.

## Data Model

The PubMiner application relies on a simple DynamoDB table as the data persistence solution. This table will contain PMCIDs and the related extracted information.

This choice of a DynamoDB database was primarily made for ease of use by other AWS elements in this project. In particular, it will integrate smoothly with an AWS Lambda function which will be the process by which updates are made to the database.

Due to DynamoDB's schema-less table design and flexibility in adding data elements to existing items, the table structure is as follows:



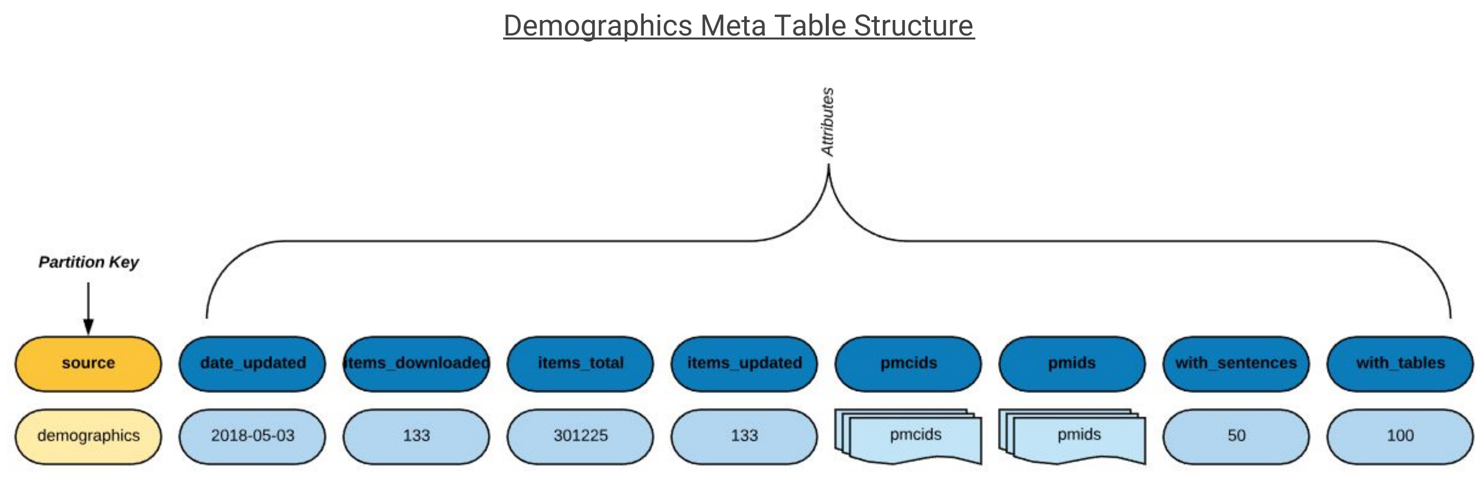


*Figure 15. DynamoDB tables demographics and demographics_meta definition*

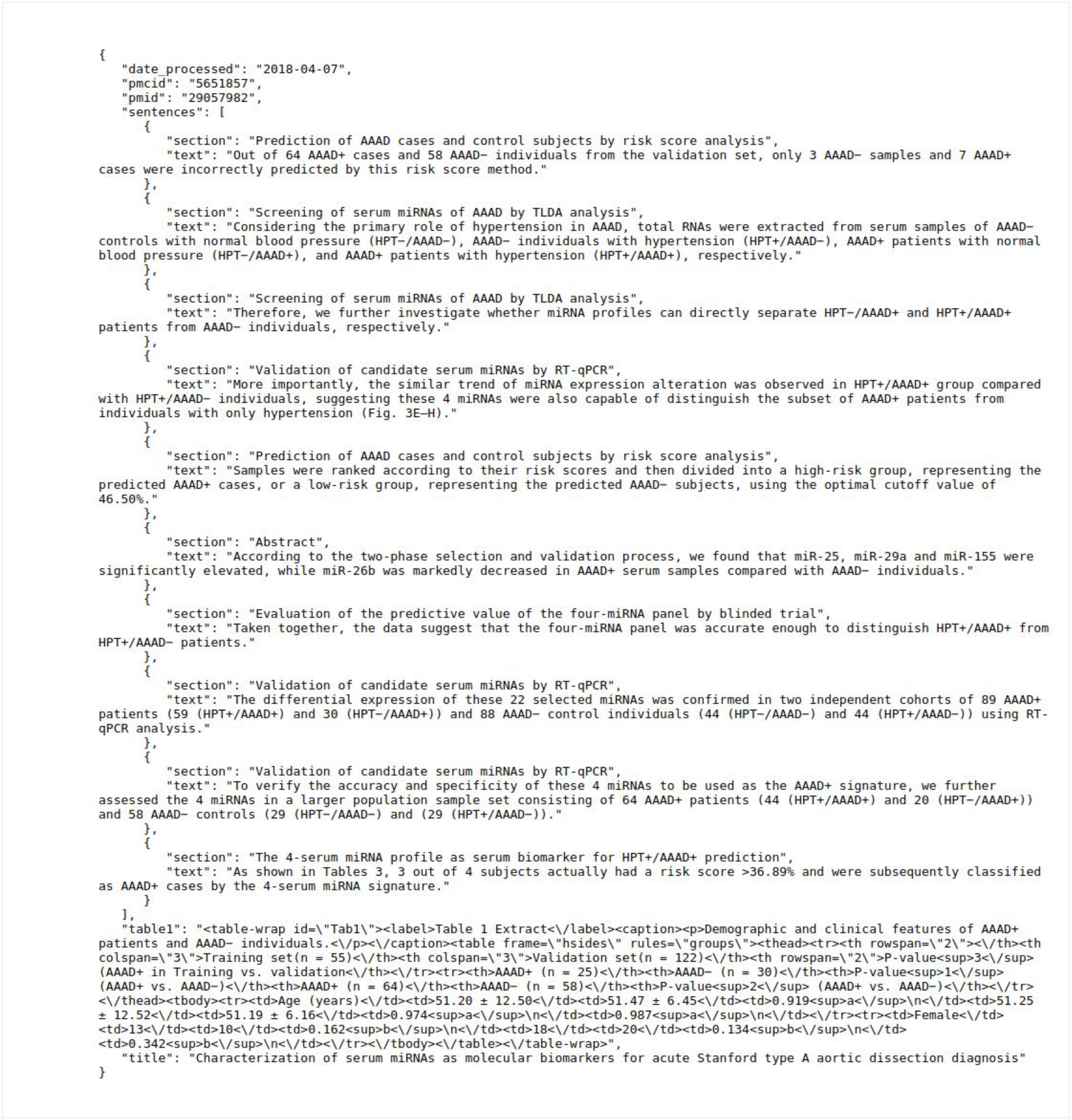
Here we define that each item of the *demographics* table has a primary hash key PMCID that is a string object. Each article item will have a set of associates attributes, including PMID, date_processed, title, a set of sentences, and an html version of a table.

An additional *demographics_meta* table describes the results of each batch processing pipeline so that the web application can display this information to the user.

Example Item Structure:

Each item stores the demographic sentences, additional information about sentences, the extracted HTML of the relevant section of Table 1 when available, and other metadata such as the date the article was processed. Note that the sentences are stored in order from most to least likely to contain demographic information. These are displayed in the web application first grouped by section name and then ordered within each section. Figure 16, below, shows a sample item:

<u>Sample DynamoDB Object</u>



```
{
   "date_processed": "2018-04-07",
   "pmcid": "5651857",
   "pmid": "29057982",
   "sentences": [
      {
         "section": "Prediction of AAAD cases and control subjects by risk score analysis",
         "text": "Out of 64 AAAD+ cases and 58 AAAD− individuals from the validation set, only 3 AAAD− samples and 7 AAAD+
cases were incorrectly predicted by this risk score method."
      },
      {
         "section": "Screening of serum miRNAs of AAAD by TLDA analysis",
         "text": "Considering the primary role of hypertension in AAAD, total RNAs were extracted from serum samples of AAAD−
controls with normal blood pressure (HPT−/AAAD−), AAAD− individuals with hypertension (HPT+/AAAD−), AAAD+ patients with normal
blood pressure (HPT−/AAAD+), and AAAD+ patients with hypertension (HPT+/AAAD+), respectively."
      },
      {
         "section": "Screening of serum miRNAs of AAAD by TLDA analysis",
         "text": "Therefore, we further investigate whether miRNA profiles can directly separate HPT−/AAAD+ and HPT+/AAAD+
patients from AAAD− individuals, respectively."
      },
      {
         "section": "Validation of candidate serum miRNAs by RT-qPCR",
         "text": "More importantly, the similar trend of miRNA expression alteration was observed in HPT+/AAAD+ group compared
with HPT+/AAAD− individuals, suggesting these 4 miRNAs were also capable of distinguish the subset of AAAD+ patients from
individuals with only hypertension (Fig. 3E–H)."
      },
      {
         "section": "Prediction of AAAD cases and control subjects by risk score analysis",
         "text": "Samples were ranked according to their risk scores and then divided into a high-risk group, representing the
predicted AAAD+ cases, or a low-risk group, representing the predicted AAAD− subjects, using the optimal cutoff value of
46.50%."
      },
      {
         "section": "Abstract",
         "text": "According to the two-phase selection and validation process, we found that miR-25, miR-29a and miR-155 were
significantly elevated, while miR-26b was markedly decreased in AAAD+ serum samples compared with AAAD− individuals."
      },
      {
         "section": "Evaluation of the predictive value of the four-miRNA panel by blinded trial",
         "text": "Taken together, the data suggest that the four-miRNA panel was accurate enough to distinguish HPT+/AAAD+ from
HPT+/AAAD− patients."
      },
      {
         "section": "Validation of candidate serum miRNAs by RT-qPCR",
         "text": "The differential expression of these 22 selected miRNAs was confirmed in two independent cohorts of 89 AAAD+
patients (59 (HPT+/AAAD+) and 30 (HPT−/AAAD+)) and 88 AAAD− control individuals (44 (HPT−/AAAD−) and 44 (HPT+/AAAD−)) using RT-
qPCR analysis."
      },
      {
         "section": "Validation of candidate serum miRNAs by RT-qPCR",
         "text": "To verify the accuracy and specificity of these 4 miRNAs to be used as the AAAD+ signature, we further
assessed the 4 miRNAs in a larger population sample set consisting of 64 AAAD+ patients (44 (HPT+/AAAD+) and 20 (HPT−/AAAD+))
and 58 AAAD− controls (29 (HPT−/AAAD−) and (29 (HPT+/AAAD−))."
      },
      {
         "section": "The 4-serum miRNA profile as serum biomarker for HPT+/AAAD+ prediction",
         "text": "As shown in Tables 3, 3 out of 4 subjects actually had a risk score >36.89% and were subsequently classified
as AAAD+ cases by the 4-serum miRNA signature."
      }
   ],
   "table1": "<table-wrap id=\"Tab1\"><label>Table 1 Extract<\/label><caption><p>Demographic and clinical features of AAAD+
patients and AAAD− individuals.<\/p><\/caption><table frame=\"hsides\" rules=\"groups\"><thead><tr><th rowspan=\"2\"><\/th><th
colspan=\"3\">Training set(n = 55)<\/th><th colspan=\"3\">Validation set(n = 122)<\/th><th rowspan=\"2\">P-value<sup>3<\/sup>
(AAAD+ in Training vs. validation<\/th><\/tr><tr><th>AAAD+ (n = 25)<\/th><th>AAAD− (n = 30)<\/th><th>P-value<sup>1<\/sup>
(AAAD+ vs. AAAD−)<\/th><th>AAAD+ (n = 64)<\/th><th>AAAD− (n = 58)<\/th><th>P-value<sup>2<\/sup> (AAAD+ vs. AAAD−)<\/th><\/tr>
<\/thead><tbody><tr><td>Age (years)<\/td><td>51.20 ± 12.50<\/td><td>51.47 ± 6.45<\/td><td>0.919<sup>a<\/sup>\n<\/td><td>51.25
± 12.52<\/td><td>51.19 ± 6.16<\/td><td>0.974<sup>a<\/sup>\n<\/td><td>0.987<sup>a<\/sup>\n<\/td><\/tr><tr><td>Female<\/td>
<td>13<\/td><td>10<\/td><td>0.162<sup>b<\/sup>\n<\/td><td>18<\/td><td>20<\/td><td>0.134<sup>b<\/sup>\n<\/td>
<td>0.342<sup>b<\/sup>\n<\/td><\/tr><\/tbody><\/table><\/table-wrap>",
   "title": "Characterization of serum miRNAs as molecular biomarkers for acute Stanford type A aortic dissection diagnosis"
}
```

*Figure 16. Example database item*

## PubMiner Network Architecture

PubMiner resides on AWS infrastructure, and while the DynamoDB databases and AWS Lambda functions are serverless, the two Web Server instances, and the Sentence Miner instance are within a VPC tailored for the application, and an Elastic Load Balancer and Amazon Route 53 domain (pubminer.org) complete the network architecture. The complete network architecture is illustrated in figure 17.

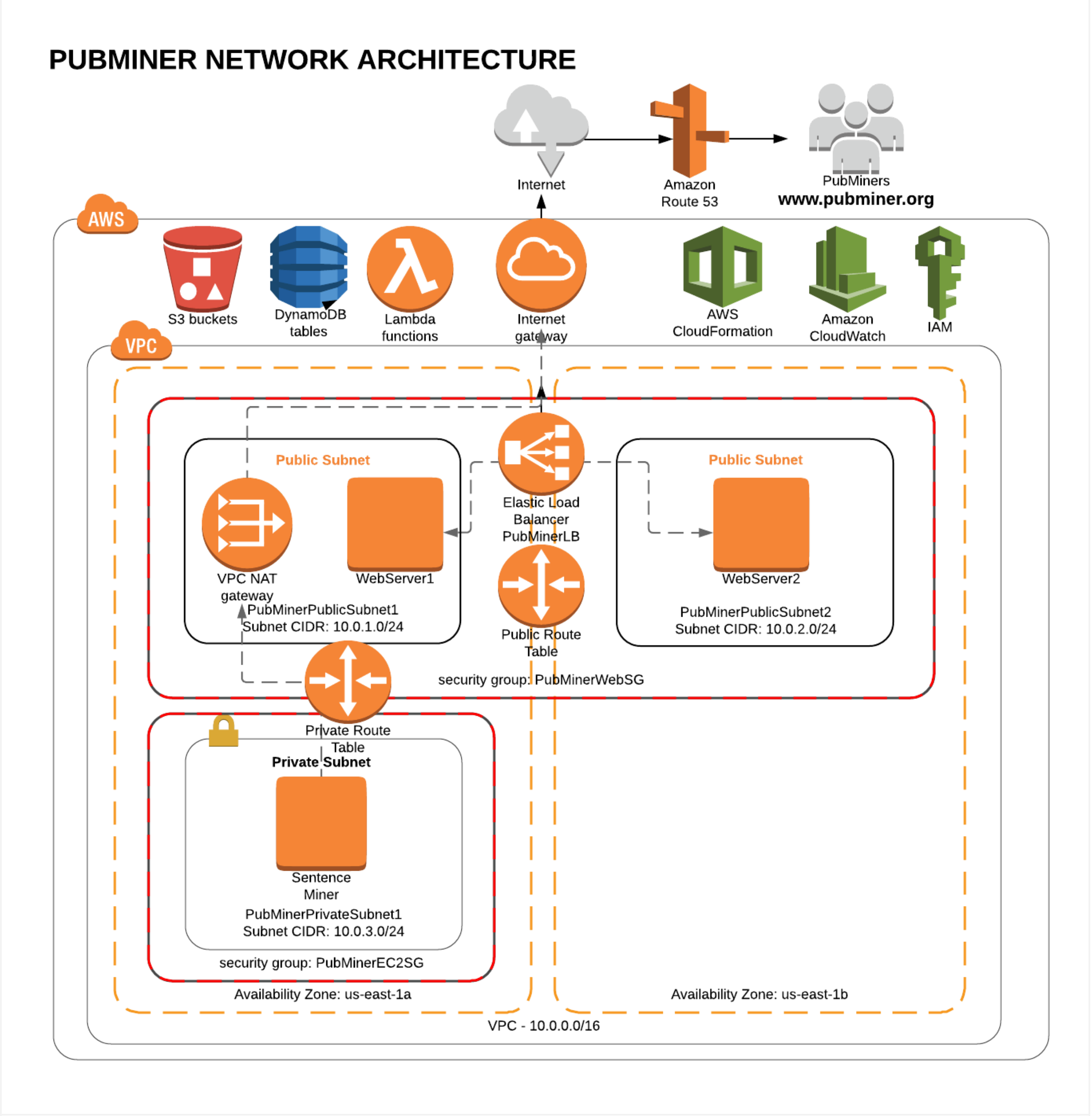


*Figure 17. PubMiner network architecture on AWS*

The VPC has two public subnets, and one private subnet. The former share a web security group with open HTTP access, and the latter has a scaled back security group with only SSH access (through a NAT gateway). PubMinerPublicSubnet1 and PubMinerPrivateSubnet1 are both in the us-east-1a Availability Zone, and PubMinerPublicSubnet2 is in the us-east-1b Availability Zone. Users are routed through the Elastic Load Balancer PubMinerLB to one of the WebServer instances, each of which resides in its own public subnet. While internet traffic from the Web Server instances is routed through a public route table, for the private subnet PubMinerPrivateSubnet1, traffic is constrained to a private route table that routes through the VPC NAT Gateway located in PubMinerPublicSubnet1. Note, that the Sentence Miner instance is not permanent, it launches on demand.

The other AWS services used are not located in the VPC. They include two S3 buckets (pubmedcentral_oa and pubminer_upload), the Dynamodb tables (demographics and demographics_meta), the AWS Lambda functions created by CloudFormation, as well as the logs and Lambda function triggers in Amazon CloudWatch.

## Web Application

The web application is written as a three-tiered architecture with a front end, web service, and database.

### Application Architecture And Technology

The web application architecture is illustrated alongside the overall application architecture along with the core technologies for each component in figure 18.

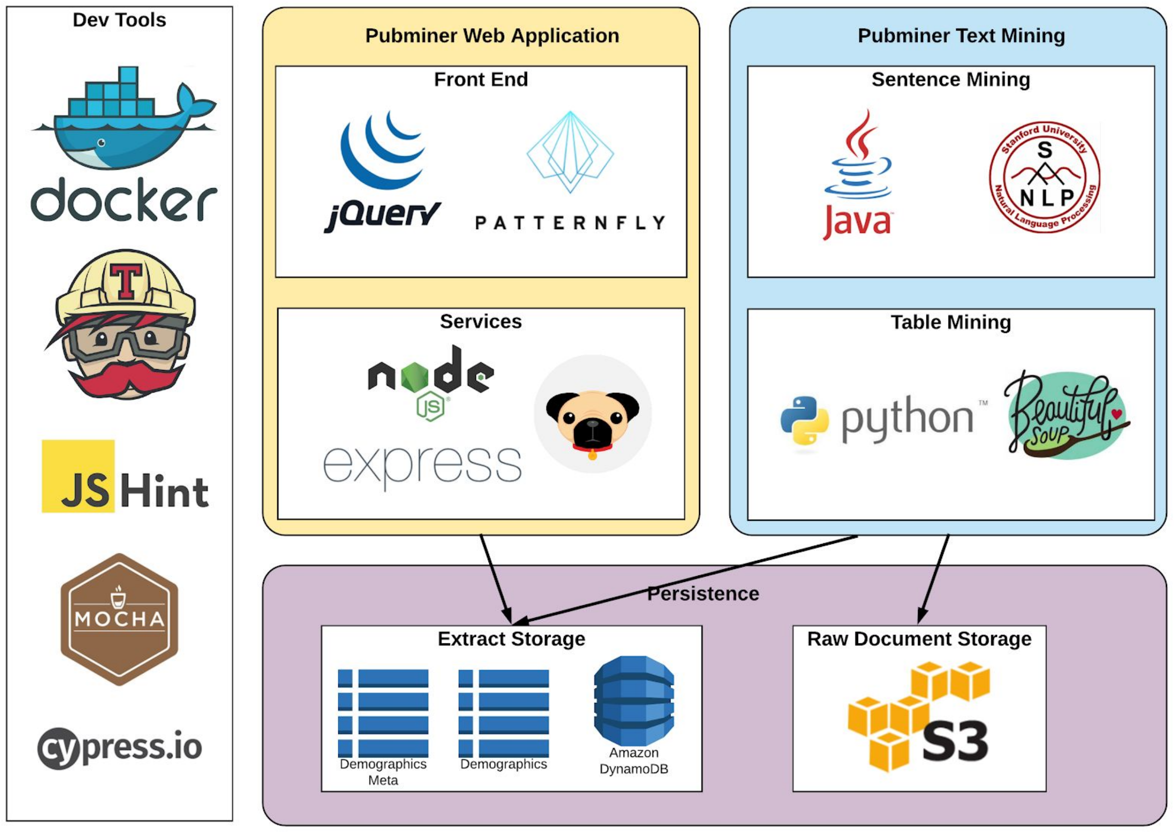


*Figure 18. PubMiner web application architecture and technologies*

The web application component of PubMiner is written in JavaScript using the NodeJS runtime. Several factors contributed to this choice of technology. Chief among them was the fact that most team members had some level of experience with JavaScript. This meant that most team members could contribute to development and that the team could avoid a situation where a sole expert might become a bottleneck during development. The particular problem domain of the PubMiner's web application deals with orchestrating calls to various service endpoints and transforming and combining that data for presentation to the UI. This use case lends itself very well to a dynamically-typed language that natively supports JSON. The choice of language and runtime allowed developers to forego much of the effort normally required to encode and decode data in statically-typed contexts, and the large community around NodeJS made the process of finding answers to the issues encountered during development easier.

## Software Architecture

The web application comprises three core of software services. Two of these services represent the data sources for the system. The PubMedService provides high-level APIs for calling NCBI's e-utilties APIs and standardizing their response formats. The DemographicService wraps access to PubMiner's own demographic data stored in AWS DynamoDB. These two services are independent and have no knowledge of each other. The third service, the PubMinerService, furnishes a set of operations that map almost directly to HTTP endpoints exposed by the application. The PubMinerService orchestrates calls to the lower level services to provide fundamental application features like searching and showing search results. Figure 19 highlights the web application's software architecture.

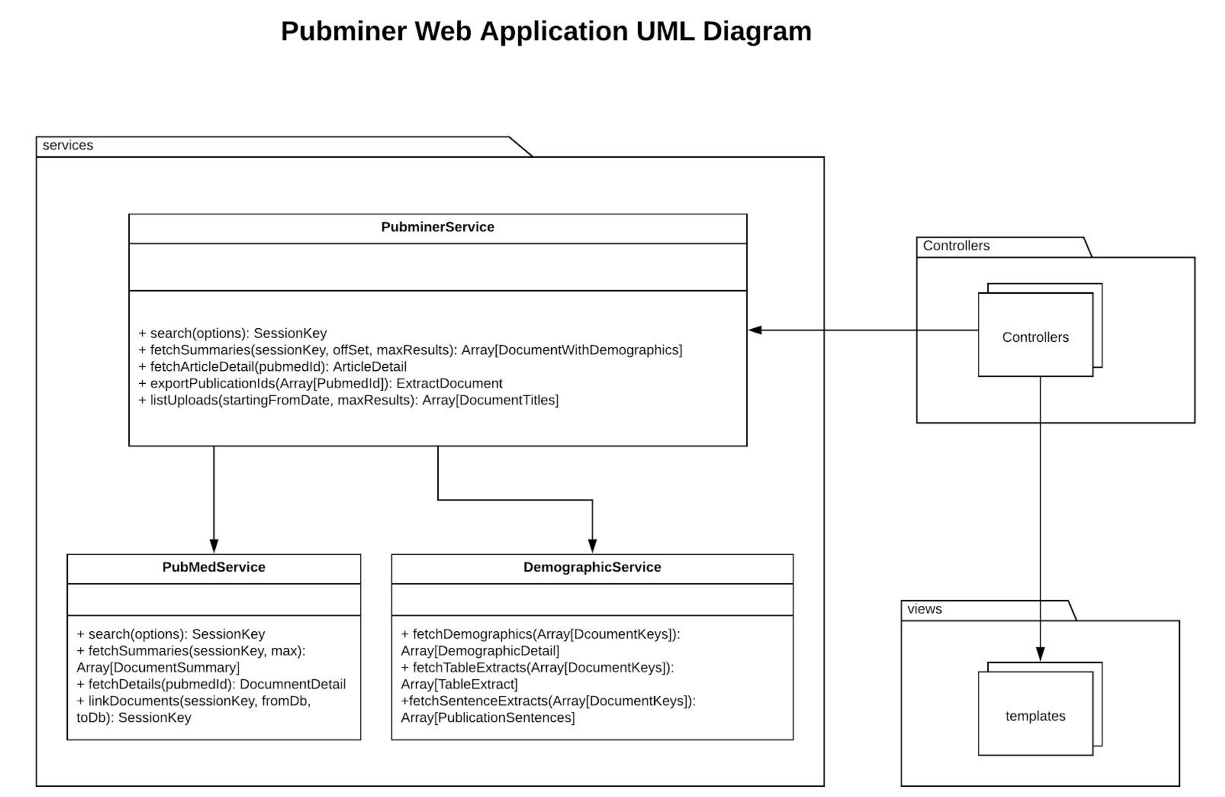
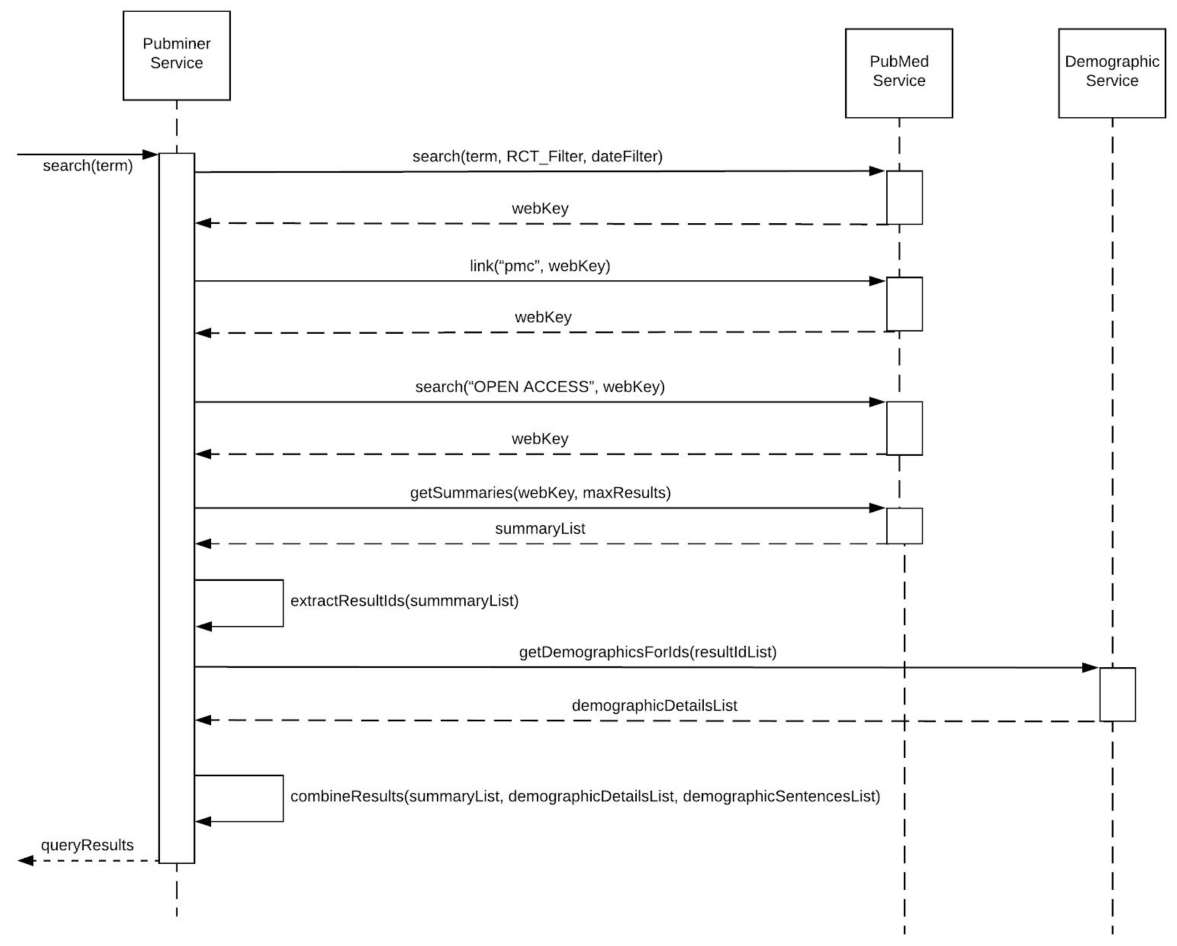

Pubminer Web Application UML Diagram
services
PubminerService
+ search(options): SessionKey
+ fetchSummaries(sessionKey, offSet, maxResults): Array[DocumentWithDemographics]
+ fetchArticleDetail(pubmedId): ArticleDetail
+ exportPublicationIds(Array[PubmedId]): ExtractDocument
+ listUploads(startingFromDate, maxResults): Array[DocumentTitles]
PubMedService
+ search(options): SessionKey
+ fetchSummaries(sessionKey, max): Array[DocumentSummary]
+ fetchDetails(pubmedId): DocumnentDetail
+ linkDocuments(sessionKey, fromDb, toDb): SessionKey
DemographicService
+ fetchDemographics(Array[DcoumentKeys]): Array[DemographicDetail]
+ fetchTableExtracts(Array[DocumentKeys]): Array[TableExtract]
+fetchSentenceExtracts(Array[DocumentKeys]): Array[PublicationSentences]
Controllers
Controllers
views
templates

*Figure 19. PubMiner web application UML diagram*

## Data Enrichment Process

PubMiner provides the enriched search results to users by combining data from two different data sources. PubMiner first performs a series of search, filter, and fetch operations using the NCBI e-utilities APIs. The result of these calls is an initial list of summary results that are likely to be in PubMiner's database since the search terms are enriched with filters (see figure 1) and limited to open-access articles. Once PubMiner has received the first set of results from NCBI, it pulls the PMCIDs from the response and queries the demographic database for matching records. The results from the demographic database, which include sentences and possibly table data, are merged with the summary results received from NCBI. Those results are then returned to the UI. Figure 20 illustrates this process in detail.

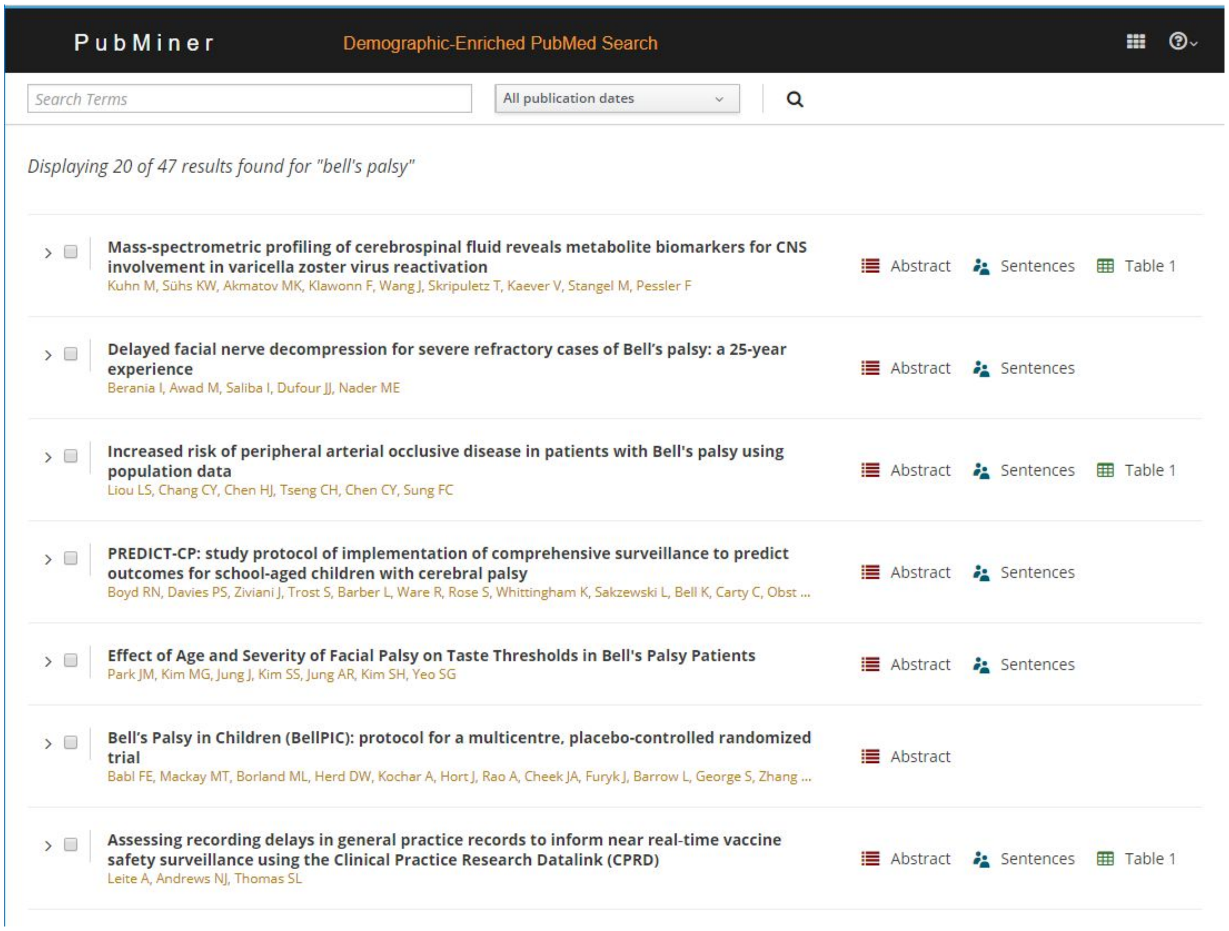


Pubminer Service
PubMed Service
Demographic Service
search(term)
search(term, RCT_Filter, dateFilter)
webKey
link("pmc", webKey)
webKey
search("OPEN ACCESS", webKey)
webKey
getSummaries(webKey, maxResults)
summaryList
extractResultIds(summmaryList)
getDemographicsForIds(resultIdList)
demographicDetailsList
combineResults(summaryList, demographicDetailsList, demographicSentencesList)
queryResults

*Figure 20. Data enrichment process sequence diagram*

## Front End

The PubMiner user interface is straightforward. The user is presented with an input field where search terms may be entered. When the user presses the search button, search results are returned and rendered on the page.  Additional information, including demographic details, can be viewed by expanding a result row. Individual result rows can be checked to build a list of items for export to a CSV file. The initial result display contains 20 items, and additional items can be loaded.


PubMiner Demographic-Enriched PubMed Search

Search Terms | All publication dates

Displaying 20 of 47 results found for "bell's palsy"

**Mass-spectrometric profiling of cerebrospinal fluid reveals metabolite biomarkers for CNS involvement in varicella zoster virus reactivation**
Kuhn M, Sühs KW, Akmatov MK, Klawonn F, Wang J, Skripuletz T, Kaever V, Stangel M, Pessler F
Abstract | Sentences | Table 1

**Delayed facial nerve decompression for severe refractory cases of Bell's palsy: a 25-year experience**
Berania I, Awad M, Saliba I, Dufour JJ, Nader ME
Abstract | Sentences

**Increased risk of peripheral arterial occlusive disease in patients with Bell's palsy using population data**
Liou LS, Chang CY, Chen HJ, Tseng CH, Chen CY, Sung FC
Abstract | Sentences | Table 1

**PREDICT-CP: study protocol of implementation of comprehensive surveillance to predict outcomes for school-aged children with cerebral palsy**
Boyd RN, Davies PS, Ziviani J, Trost S, Barber L, Ware R, Rose S, Whittingham K, Sakzewski L, Bell K, Carty C, Obst ...
Abstract | Sentences

**Effect of Age and Severity of Facial Palsy on Taste Thresholds in Bell's Palsy Patients**
Park JM, Kim MG, Jung J, Kim SS, Jung AR, Kim SH, Yeo SG
Abstract | Sentences

**Bell's Palsy in Children (BellPIC): protocol for a multicentre, placebo-controlled randomized trial**
Babl FE, Mackay MT, Borland ML, Herd DW, Kochar A, Hort J, Rao A, Cheek JA, Furyk J, Barrow L, George S, Zhang ...
Abstract

**Assessing recording delays in general practice records to inform near real-time vaccine safety surveillance using the Clinical Practice Research Datalink (CPRD)**
Leite A, Andrews NJ, Thomas SL
Abstract | Sentences | Table 1

*Figure 21. PubMiner Search Results Page*

## Search

Values entered in the search terms field are combined with a filter designed to find only randomized clinical trials with human participants. Using NCBI's E-utilities API, the PubMed database is searched and a result set consisting of PubMed article ids is stored on the NCBI server. The search result provides web environment (webenv) and query key values that can be used to access the stored results and provide the stored article ids as input to other E-utilities. We implemented a search query string to filter for clinical trials after reviewing literature by Glanville, et al [1] the Cochrane Work website[2]. The filter we chose to use is:

```
((randomized+controlled+trial[pt])+OR+(controlled+clinical+trial[pt])+OR+(randomized[tiab]+OR+randomised[tiab])+OR+(placebo[tiab])+OR+(drug+therapy[sh])+OR+(randomly[tiab])+OR+(trial[tiab])+OR+(groups[tiab]))+NOT+(animals[mh]+NOT+humans[mh])
```

[1] Glanville, J. M., Lefebvre, C., Miles, J. N. V., & Camosso-Stefinovic, J. (2006), How to identify randomized controlled trials in MEDLINE: ten years on. Journal of the Medical Library Association, 94(2), 130–136.
[2] The Cochrane highly sensitive search strategies for identifying randomized trials in PubMed. (Retrieved March 2018), from Cochrane Work: http://work.cochrane.org/pubmed

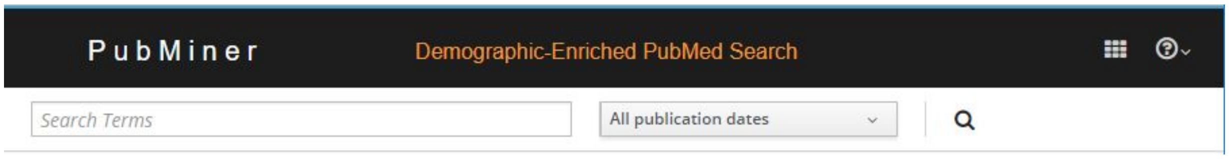


Figure 22. PubMiner Search Toolbar

## Results - Basic Display

Once the search is completed a page of twenty results is presented to the user. E-utilities are employed again to retrieve article summaries for the first twenty results ids. Additionally, the PubMiner database is queried to retrieve any demographic table or sentence data that has been retrieved for each article id.

A row is rendered for each result showing:

- an arrow that expands the row to reveal additional information about the article, including demographic data from our database.
- a checkbox that allows the user to select this row for export to a CSV file.
- the article title
- the article's authors
- demographic tags to give a quick visual indicator about what demographic data may be available for this article. Note: the exact nature of the tags has not been finalized yet. Figure 20 shows an early model where we presented age, gender and race information. The final tags will likely be much broader, e.g. "Table 1 data available".

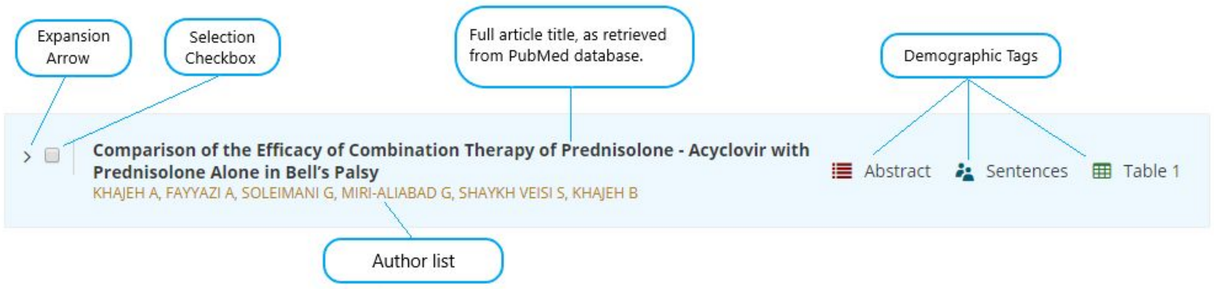
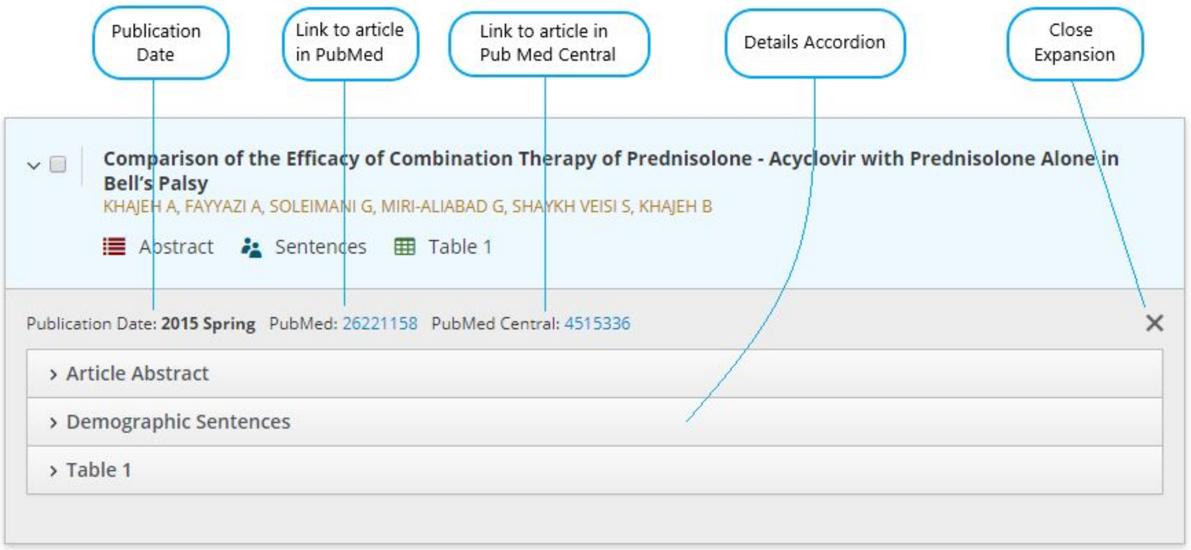

Expansion Arrow
Selection Checkbox
Full article title, as retrieved from PubMed database.
Demographic Tags
Comparison of the Efficacy of Combination Therapy of Prednisolone - Acyclovir with Prednisolone Alone in Bell's Palsy
KHAJEH A, FAYYAZI A, SOLEIMANI G, MIRI-ALIABAD G, SHAYKH VEISI S, KHAJEH B
Abstract
Sentences
Table 1
Author list

Figure 23. PubMiner Result Row - Normal

## Results - Expanded Display

The user may click on  the row's arrow to display expanded information about the article, including information that has been pulled from the table and sentence mining database. The expansion area includes:

- the article's publication date
- a link to the article in PubMed
- a link to the article in PubMed Central, if available
- an accordion control that can be expanded to show demographic details

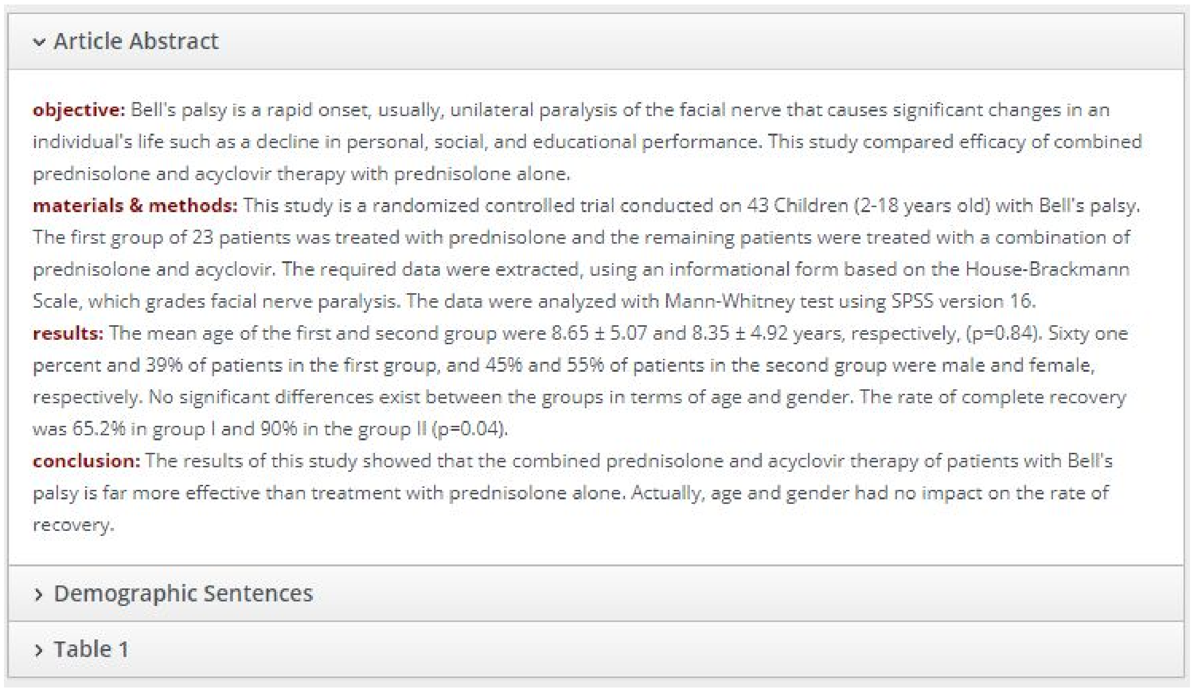


Comparison of the Efficacy of Combination Therapy of Prednisolone - Acyclovir with Prednisolone Alone in Bell's Palsy
KHAJEH A, FAYYAZI A, SOLEIMANI G, MIRI-ALIABAD G, SHAYKH VEISI S, KHAJEH B

Publication Date: **2015 Spring** PubMed: 26221158 PubMed Central: 4515336

*Figure 24. PubMiner Result Row - Expanded*

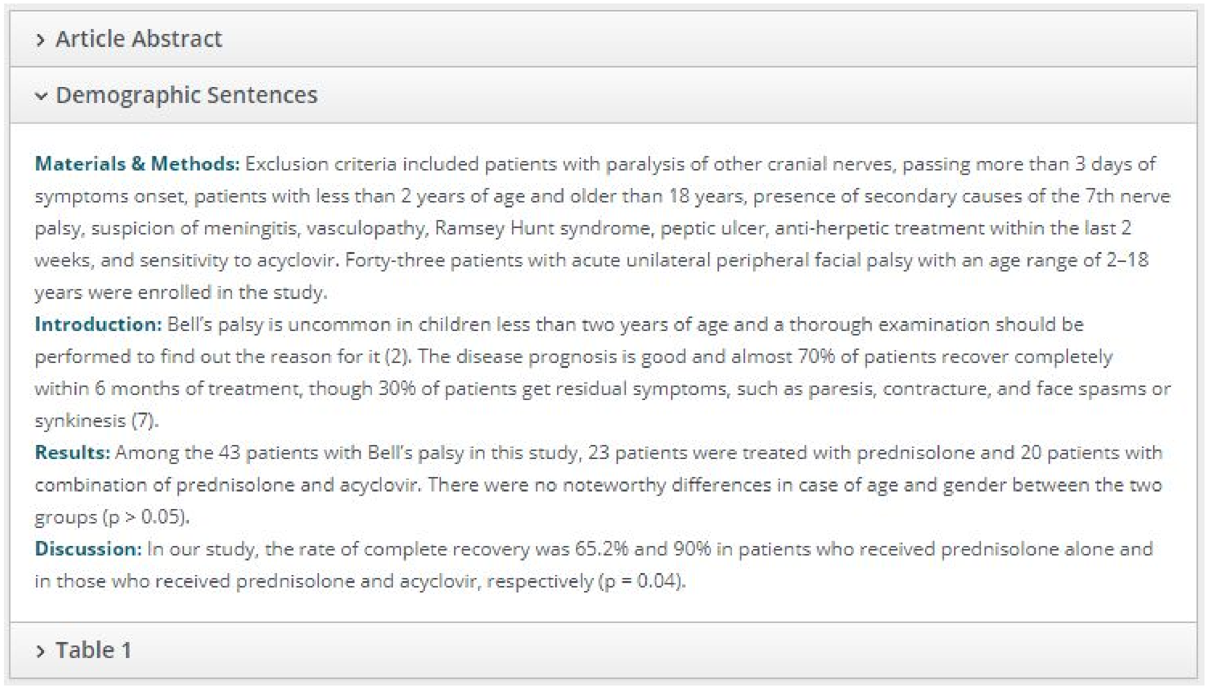
The details accordion can be expanded to show three sections that may contain demographic data.  The article abstract section displays the abstract as retrieved from the PubMed database.



**objective:** Bell's palsy is a rapid onset, usually, unilateral paralysis of the facial nerve that causes significant changes in an individual's life such as a decline in personal, social, and educational performance. This study compared efficacy of combined prednisolone and acyclovir therapy with prednisolone alone.
**materials & methods:** This study is a randomized controlled trial conducted on 43 Children (2-18 years old) with Bell's palsy. The first group of 23 patients was treated with prednisolone and the remaining patients were treated with a combination of prednisolone and acyclovir. The required data were extracted, using an informational form based on the House-Brackmann Scale, which grades facial nerve paralysis. The data were analyzed with Mann-Whitney test using SPSS version 16.
**results:** The mean age of the first and second group were 8.65 ± 5.07 and 8.35 ± 4.92 years, respectively, (p=0.84). Sixty one percent and 39% of patients in the first group, and 45% and 55% of patients in the second group were male and female, respectively. No significant differences exist between the groups in terms of age and gender. The rate of complete recovery was 65.2% in group I and 90% in the group II (p=0.04).
**conclusion:** The results of this study showed that the combined prednisolone and acyclovir therapy of patients with Bell's palsy is far more effective than treatment with prednisolone alone. Actually, age and gender had no impact on the rate of recovery.

*Figure 24a. PubMiner Result Row - Accordion - Article Abstract*

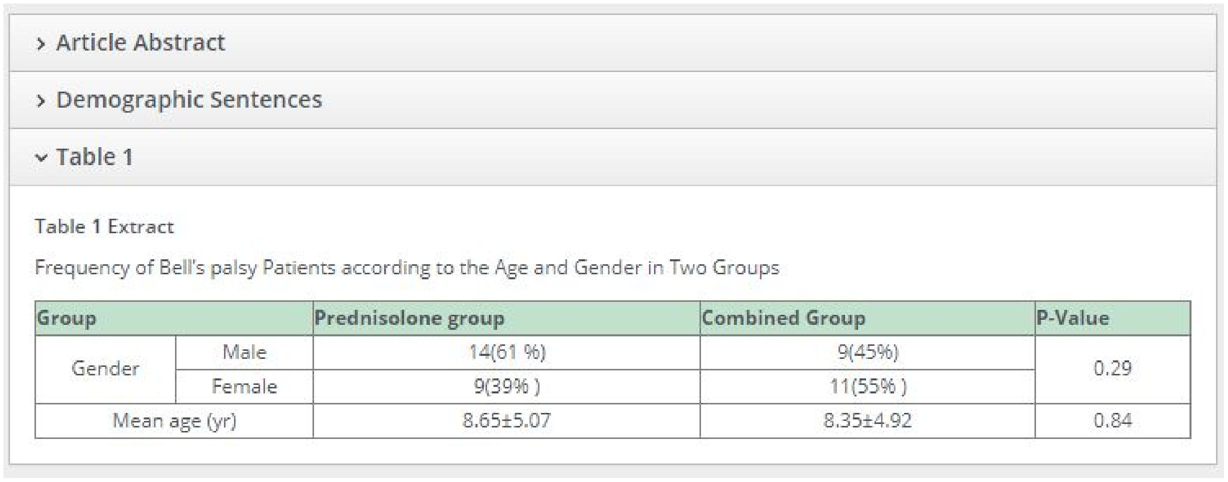
The middle section of the accordion shows demographic sentences that have been mined from the article's full text. The section where the sentence appeared is shown and highlighted.



**Materials & Methods:** Exclusion criteria included patients with paralysis of other cranial nerves, passing more than 3 days of symptoms onset, patients with less than 2 years of age and older than 18 years, presence of secondary causes of the 7th nerve palsy, suspicion of meningitis, vasculopathy, Ramsey Hunt syndrome, peptic ulcer, anti-herpetic treatment within the last 2 weeks, and sensitivity to acyclovir. Forty-three patients with acute unilateral peripheral facial palsy with an age range of 2–18 years were enrolled in the study.
**Introduction:** Bell's palsy is uncommon in children less than two years of age and a thorough examination should be performed to find out the reason for it (2). The disease prognosis is good and almost 70% of patients recover completely within 6 months of treatment, though 30% of patients get residual symptoms, such as paresis, contracture, and face spasms or synkinesis (7).
**Results:** Among the 43 patients with Bell's palsy in this study, 23 patients were treated with prednisolone and 20 patients with combination of prednisolone and acyclovir. There were no noteworthy differences in case of age and gender between the two groups (p > 0.05).
**Discussion:** In our study, the rate of complete recovery was 65.2% and 90% in patients who received prednisolone alone and in those who received prednisolone and acyclovir, respectively (p = 0.04).

*Figure 24b. PubMiner Result Row - Accordion - Demographic Sentences*

The bottom section of the accordion displays excerpts from the article's table 1, if one was found. Only sections determined to contain demographic data are included in the table excerpt.



› Article Abstract

› Demographic Sentences

⌄ Table 1

Table 1 Extract

Frequency of Bell's palsy Patients according to the Age and Gender in Two Groups

| Group | | Prednisolone group | Combined Group | P-Value |
|---|---|---|---|---|
| Gender | Male | 14(61 %) | 9(45%) | 0.29 |
| | Female | 9(39% ) | 11(55% ) | |
| Mean age (yr) | | 8.65±5.07 | 8.35±4.92 | 0.84 |

*Figure 24c. PubMiner Result Row - Accordion - Table 1*

## User Interface Design

**PatternFly** (http://www.patternfly.org/)

PubMiner takes advantage of the PatternFly library of visual design elements. PatternFly is built using Bootstrap styles and provides a rich library of ready-to-use design elements. The PubMiner user interface uses the "List View With Expanding Rows" pattern to display results returned from the data service.

**Pug** (https://github.com/pugjs/pug)

Pages and page segments are rendered using the Pug template engine. Visual elements are designed using Pug templates. The Pug rendering engine converts the templates and data into html on the back end, and the html is delivered to the browser for rendering. While it takes some time to get used to the Pug syntax, it makes certain tasks, like rendering repetitive elements, much easier.

## 2. Testing the Results

We performed both unit and integration testing on each of the components of the PubMiner system, including unit and integration testing on the web application, and testing on the table and sentence mining processes as well as integration testing on the data pipeline as a whole.

### Process to Construct the Article Database

#### Testing of AWS Lambda functions

The AWS lambda functions integrate a small subset of the extensive Python code and functions which have been used in the data discovery and wrangling stages of the project, and which have been tested during that work with canned XML and HTML input files. The AWS Lambda functions are short and contain no classes. For example, the process of narrowing search results down to the subset of target articles depends (only) on several functions in Python that have been integrated in a single Lambda function, and which have been used extensively in the development work. The Python code for this particular Lambda function is, including comments, less than 150 lines.

The testing of the AWS Lambda functions proceeded in three ways. AWS Lambda has a built-in "Test" feature which allows testing of trigger events, and visualizing the output of the function from test data in the event. The test data can be constructed in order to test the full functionality of the code (see Appendix for results). Second, print statements were extensively used to ensure that the output of the Lambda functions was producing both the desired output for the function and the required stream or trigger information for subsequent Lambda functions. For the latter, the AWS CloudWatch logs were used to tailor the event information output from the functions. Third, canned XML and HTML input files were used to trigger the entire process, and the CloudWatch logs, as well as the resulting database records were inspected to ensure correct process flow and output.

#### Sentence Mining

This component of the project is written in Java and JUnit is used to test this component. We fed in a set of PMC articles to assert that our program produces the expected results. The set of PMC articles and demographic sentences came from another tool, robotreviewer.net, which does population annotations of randomized clinical trial articles. The test result consistently shows that our tool discovers the correct demographic sentences and does a better job than robotreviewer.net.

Table Mining

The table truncation process depends on Python functions with sequential parsing with regular expressions using lxml and BeautifulSoup, and these functions were tested using the BeautifulSoup testing classes that are provided at crummy.com , which have been created by one of the main contributors of BeautifulSoup to test the package (see Appendix for results). In addition, we used about two dozen canned XML and HTML input files with various table formats and containing the terms to test: (1) the construction of trees and the fetching of tags; (2) the regular expression rules; (3) proximity rules for determining siblings; and (4) the truncation results. For example, the order and position in the table of key terms such as “gender” or “hispanic” was varied in the table included in the input XML file.

The resulting truncated tables were inspected to ensure that the logic in the extraction of table elements conformed to our customer’s requirements to keep only the table structure (including the headings) and demographic information on age, sex and race/ethnicity. A example set of 1000 tables was visually inspected in the process of honing and testing the regular expression syntax used in the parsing. Our customer also provided spot checking feedback on the results of the truncation. Two especially tricky aspects were tables with heading information in the <tbody> HTML element and tables with age groups (numeric with months, years, etc.) as subrows.

### Web Application

The web application’s software is architected in a manner that maximizes the use of purely functional operations. For example, if the application combines the results of several different API calls into a single result through a series of transformations, all transformations are written as pure functions, which are then composed with the required API calls. This approach made it possible to achieve a high-level of code coverage using traditional, dependency-less unit testing, done using the Mocha testing framework.

Some code components necessarily have to interact with external services. For example, some components orchestrate calls to NCBI’s e-utils or Amazon’s DynamoDB. These components were unit tested using an HTTP mocking library called nock. The nock NodeJS library allows HTTP calls to be mocked at the network layer which allows the code forgo using dependency injection. Instead, nock can be instructructed to verify the structure of an HTTP call and return a mock response. This library helped verify that our service produces correct calls to external services.

Cypress.io was used for integration testing of the web application and user interface. Cypress.io is a testing tool that simulates user interaction with the application by running a testing process in a web browser that accesses a locally running instance of the application. It is able to simulate typical user scenarios against a running service and verify pages are rendered as expected. This tool is used to cover PubMiner’s core user flows.

Finally, Travis CI was used to automate the application's build and run its test suite for every pull request and merge into the repository. The automated build process was also used to lint the codebase using JSHint and run a code coverage tool called codecov to generate coverage reports. This provided additional confidence for code changes that were integrated into the codebase.

## 3. Development Process & Lessons Learned

### Meeting the Requirements

After experiencing initial difficulties with the requirements, adjustments were made to reflect discoveries about the structure of article data. These adjustments were made largely the week leading up to milestone 2, which had a pronounced effect on the status of milestone 2 deliverables.

After the requirements were changed to a more reasonable and possible set, progress was made at a much quicker pace. While there were difficulties early on, all of the new requirements were satisfied by milestone 3.

Below are the restated milestone 2 and 3 deliverables labeled with their associated status as of milestone 3.

| **Milestone 2 Deliverables** | | |
|---|---|---|
| **Deliverable** | **M2 Status** | **M3 Status** |
| Data has been collected and cleaned to reasonable standards for later extraction. | Complete | Complete |
| Demographic data has been extracted from articles where the data is readily available from a standardized Table 1. | Behind Schedule | Complete |
| Sentences containing demographic data has been extracted from articles. | On Schedule | Complete |
| A database and the | Complete | Complete |

| | | |
|---|---|---|
| associated data models have been set up. | | |
| Demographic data from existing articles is stored in the database. | Behind Schedule | Complete |
| A web application with various pages and a basic user interface has been set up | Complete | Complete |

| **Milestone 3 Deliverables** | | |
|---|---|---|
| **Deliverable** | **M2 Status** | **M3 Status** |
| Web application implements filtering on demographic features, including age, gender, and race. | In Backlog | Complete |
| Users are able to export search results to a .csv format. | In Backlog | Complete |
| Investigate how to construct a training set for further machine learning techniques to extend the ability to extract demographic information from articles. | Complete | Complete |
| Curate that training dataset and explore several models to test where our solution works. | Complete | Complete |
| Test our methods against an established test dataset, such as the i2b2 NLP Research Dataset. | In Backlog | Complete |

## Resource Estimates

The logic to determine the resource availability, and thus the requirement delivery schedule for milestones 2 and 3 were originally the following:

Time Assumptions

- 5 weeks for Milestone 2
- 4 weeks for Milestone 3
- Target 12.5hrs/person/week

Time Estimates

- Milestone 2
  - 5 weeks * 12.5 hours * 6 people = 375 man hour

- Milestone 3
  - 4 weeks * 12.5 hours * 6 people = 300 man hours

While the goal was to set a time budget of 12.5 hours per person per week, this was easily exceeded throughout most of the project's timeline. This was largely due to the difficulties experienced early on in the project in understanding the customer's requirements and changing them to be more reasonable.

## Risks

Risks of the PubMiner system are categorized into one of two groups: internal and external:

Internal Risks:

- The evaluation criteria for the system may be difficult to define and not concrete. Therefore there is a risk that the users cannot determine if the solution provides value-added for their problem. This was manifested as user testing by the customers in which the team received feedback.
- Extraction of data from this quantity of full-text articles may prove to be more computationally intensive than we expect.  While this remains a risk, leveraging AWS products gives the team the flexibility to adjust computational resources as needed.

External Risks:

- PubMiner will rely on existing PubMed search APIs for search capabilities. As such, a risk to the system is the deprecation of these APIs and the reliability of PubMed itself.

- On May 1, 2018, NCBI will begin enforcing the use of API keys that will offer enhanced levels of supported access to the E-utilities. After that date, any site (IP address) posting more than 3 requests per second to the E-utilities without an API key will receive an error message.

## Team Dynamic

Due to the nature of the project and the complex subject matter with which no team member had prior experience, there were many meetings, both internal and with the customers. This was in an attempt to bring clarity to the requirements and especially to try to understand the customer’s current workflow and needs.

This tendency to meet frequently and for long hours continued as the originally accepted requirements turned out to be based on a few key false assumptions about the presence and structure of data in PubMed articles. The focus shifted away from discussion geared towards understanding the requirements and shifted towards trying to overcome significant technical hurdles.

The team was split into two groups with three allocated to research and work on the table and text mining aspects. That is, these three were assigned to prepare the enhanced data that the project relies on. The other three teammates were allocated to work on the user interface, middleware, and database aspects of the project, all of which would consume the data the first group would produce.

## Lessons Learned

### Validate unknowns and identify potential blockers as early possible

The single greatest challenge of this project was wading through PubMed’s Open-Access articles in excruciating detail as we attempted to harvest information from them. We relied heavily on EBSCO’s input during the initial formation of requirements and in obtaining an understanding of what content might be easily extracted from the PubMed Open-Access articles. It was suggested that every Randomized Controlled Trial would contain a Table 1 with a standardized presentation of demographic data. Based on this assumption, the team started to formulate a design for the system and divide responsibilities. However, as we reviewed more of the data, we encountered more and more varieties of representing Table 1 demographic information: Table 1s that were PDFs or images, RCTs without a Table 1 or for which the demographic data would not be found in any table at all, or the fact that Table 1 could assume any of dozens of different formats. Indeed, after extensive review there seemed to be no standard format to Table 1 at all.

These realizations came slowly and were hard won: each case had to be encountered, manually reviewed, documented, and, at least initially, accounted for in our table mining process. The table "mining" code seemed more like table "discovery" code. It was only after a few weeks that the full complexity of the problem became clear and we realized we would have to abandon our initial ambition of providing fine-grained demographic filters on the set of Open-Access articles. In retrospect, we should have recognized that the major aspect of the project's deliverables based on the original requirements relied on an unvalidated assumption: namely, that the Table 1 data could be readily extracted and parsed into fine-grained demographic data. Having to do it all over again, we would have time-boxed an effort to validate our initial set of assumptions and changed course in the project much sooner.

### Domain-specific knowledge is crucial for data science projects

Starting any project, like PubMiner, that involves data science, requires diving into the data. Data availability, scale, and complexity are the key considerations. You do not ever know the full extent of these consideration before diving into the data. However, often domain-specific knowledge can be crucial to quickly discover the main facets of these considerations. Due to our lack of this knowledge, our team had to build cognition quickly, primarily in three areas.

First, we had to learn about the data, such as how to distinguish between articles, in particular RCTs as opposed to other articles, the key elements of the article structure, and what-was-available-where. Second, as none of us had ever used PubMed, we did not have experience with the linked databases or background in the necessary search techniques for medical terms  (e.g. MeSH terms) and the PubMed eutils API. Third, we were not aware of how important the scale (e.g. GBytes) and complexity of the data would be in implementing a text mining solution.

The lesson learned is that everything takes time: especially the data discovery, and the research on the techniques applicable to the data science problem.  For the text mining component of the project, we started by looking at using graph databases, then turned to table mining techniques on Table1 and POS tagging, and then finally to machine learning methods for document classification of RCTs. We underestimated the potential depth of each of these areas. We found that a body of researchers have been working on each problem for years. As a result, we spent a great amount of time researching methods, and manipulating the data. The key takeaway is that, sometimes, lurking under what may appear to be straightforward problems, there are interesting, but enormous, complexities.

### Thoughtfully evaluate whether to turn a prototype into a production application

While prototypes can be great for initial proofs of concept or demonstrating some functionality to customers very quickly, one should critically evaluate whether it is appropriate to convert such prototypes into production applications. Often, prototypes dispense with a number of important software engineering practices like security, proper logging, modular design, and testability. It turns out that adding these into a working prototype can be quite problematic. In our case, we went through several round of refactoring to bring our prototype’s code quality up and get it to a place where it could be easily unit tested. Given the opportunity to take a different approach, we would opt to scrap the prototype and write the service from scratch, with a proper design.

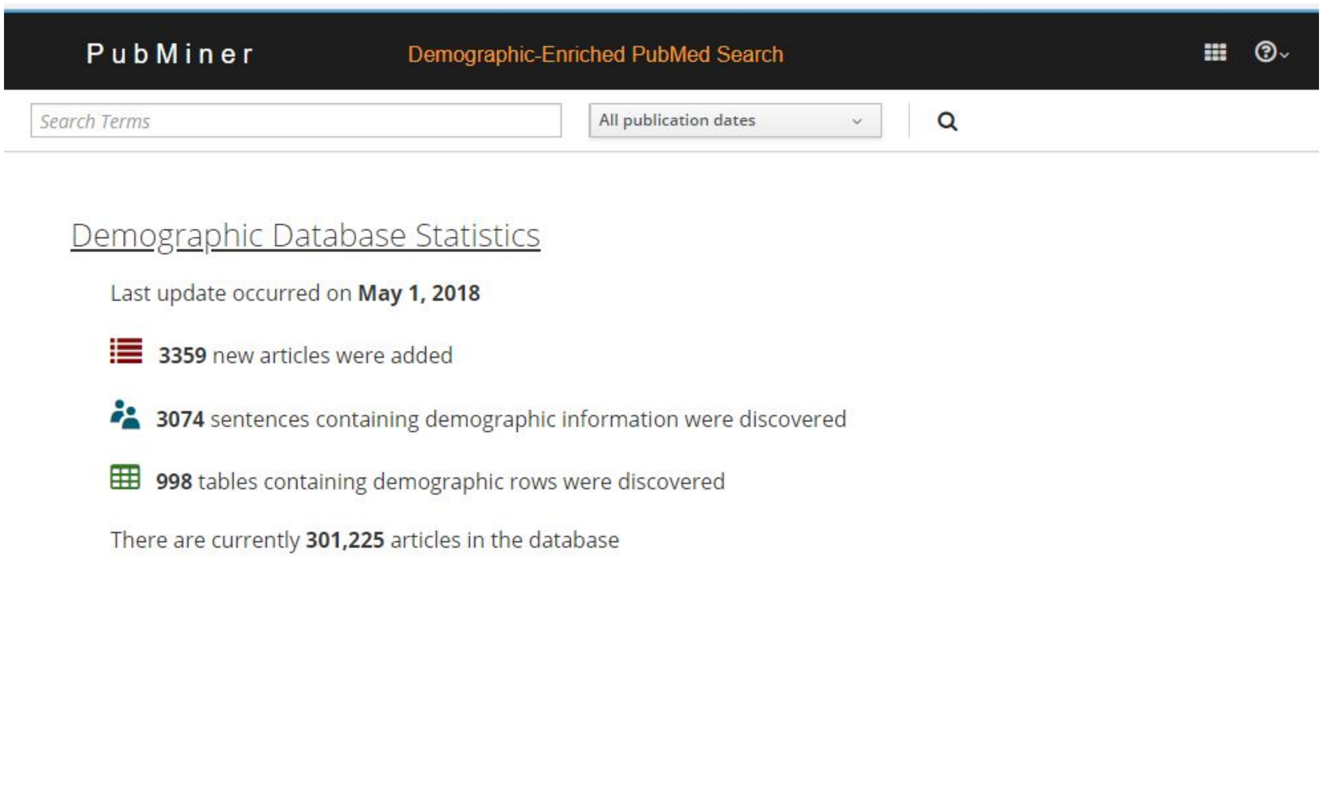
### Gauge progress early and often and adhere to a schedule

In the initial stages of the project the team focused heavily on implementing the requirements as they were originally specified. As previously discussed, this posed extensive difficulties as the requirements were set based on false assumptions but the team nonetheless continued to in our attempts. This was partly based on the belief that the requirements as originally specified were the ideal solution for the customer and that failure to meet this requirement would mean failure for the project as a whole. However, this turned out to be untrue; upon discussion with the customers there was quick agreement that the original requirement could be changed while still providing value and meeting the customer’s needs in a different way. In effect, it was far more important to provide a useful product rather than to meet the exact requirements that were agreed on. While the requirements provide a solid foundation upon which the project was built, in the end providing a working solution was the ultimate goal.

Rather than reacting to the strained and difficult progress, the team spent weeks trying to develop the product as originally envisioned, with only modest success. A few days before the second project milestone, the team met with the course instructor, and subsequently with the customer. The team and instructor developed a new set of goals that the customer approved. Going forward, the team had a clear direction and rapidly made progress on the revised goals. The time spent in weekly brainstorming meetings dropped dramatically. While the initial difficulties encountered by the team may not have been foreseeable or preventable, the project would have gone smoother had the team asked for help a couple weeks earlier.

## 4. Appendix

### Screenshots



Demographic Database Statistics

Last update occurred on **May 1, 2018**

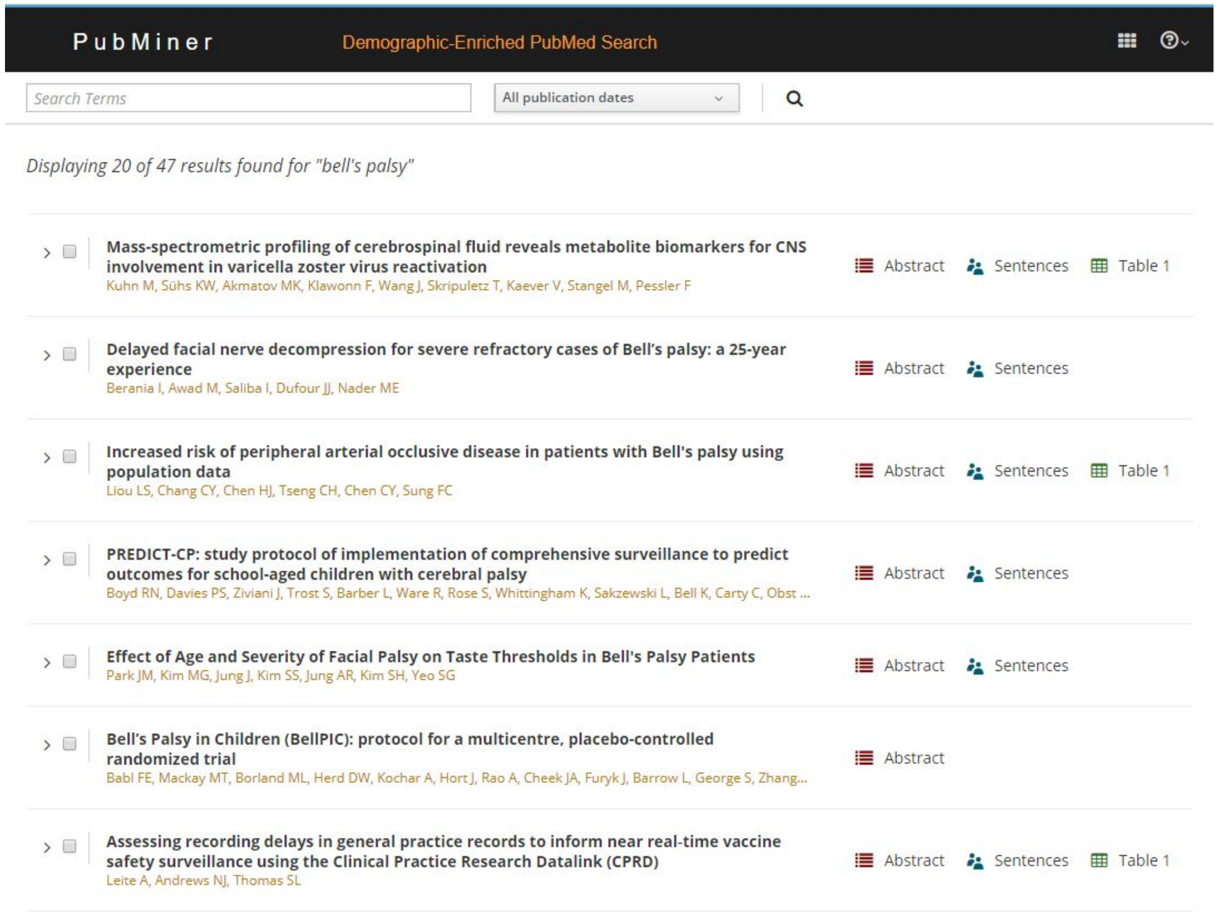
**3359** new articles were added

**3074** sentences containing demographic information were discovered

**998** tables containing demographic rows were discovered

There are currently **301,225** articles in the database

*Figure 25. PubMiner landing page, showing date of most recent update and count of articles included in the update.*


PubMiner Demographic-Enriched PubMed Search

Search Terms | All publication dates

*Displaying 20 of 47 results found for "bell's palsy"*

**Mass-spectrometric profiling of cerebrospinal fluid reveals metabolite biomarkers for CNS involvement in varicella zoster virus reactivation**
Kuhn M, Sühs KW, Akmatov MK, Klawonn F, Wang J, Skripuletz T, Kaever V, Stangel M, Pessler F
Abstract · Sentences · Table 1

**Delayed facial nerve decompression for severe refractory cases of Bell's palsy: a 25-year experience**
Berania I, Awad M, Saliba I, Dufour JJ, Nader ME
Abstract · Sentences

**Increased risk of peripheral arterial occlusive disease in patients with Bell's palsy using population data**
Liou LS, Chang CY, Chen HJ, Tseng CH, Chen CY, Sung FC
Abstract · Sentences · Table 1

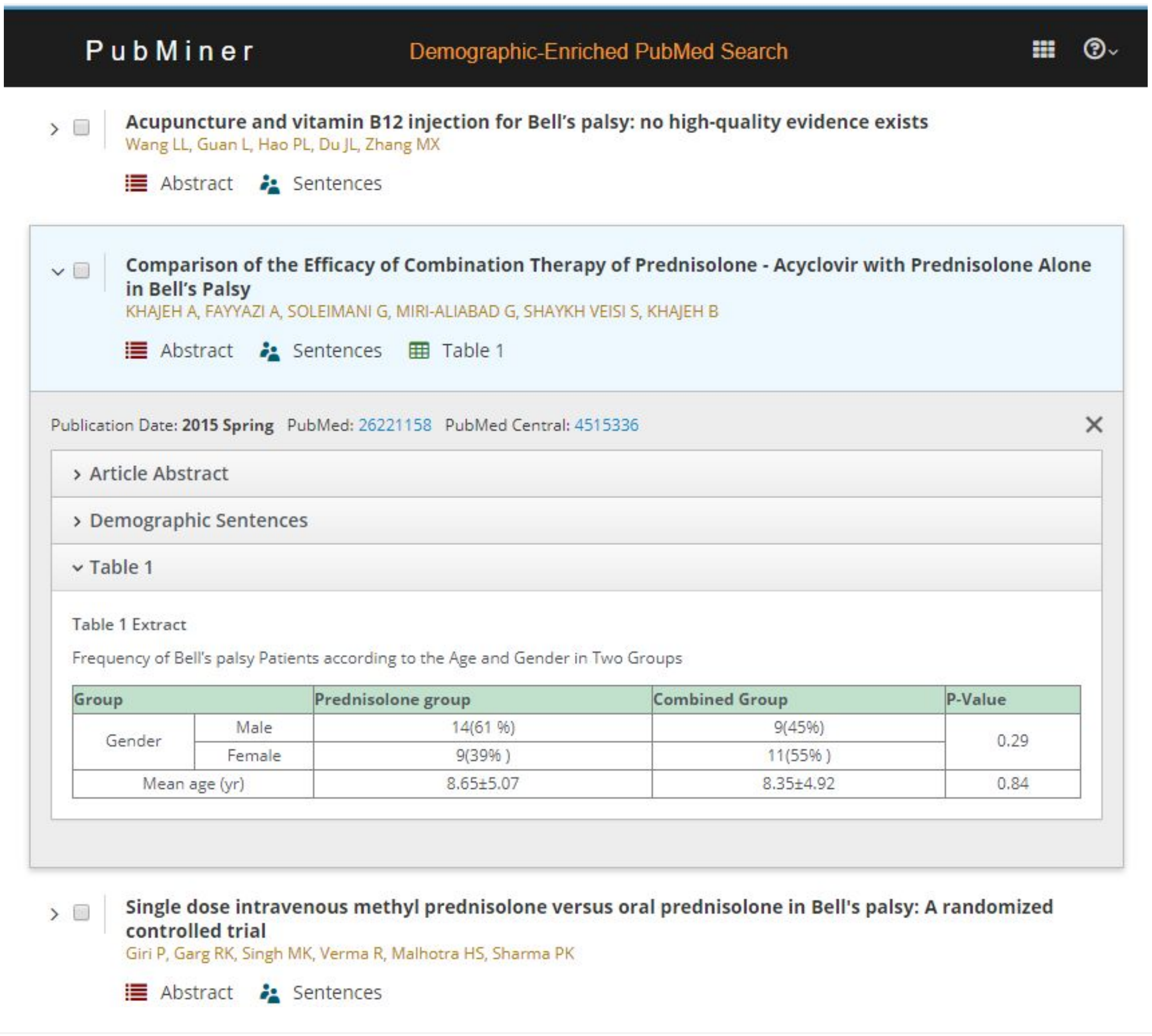
**PREDICT-CP: study protocol of implementation of comprehensive surveillance to predict outcomes for school-aged children with cerebral palsy**
Boyd RN, Davies PS, Ziviani J, Trost S, Barber L, Ware R, Rose S, Whittingham K, Sakzewski L, Bell K, Carty C, Obst ...
Abstract · Sentences

**Effect of Age and Severity of Facial Palsy on Taste Thresholds in Bell's Palsy Patients**
Park JM, Kim MG, Jung J, Kim SS, Jung AR, Kim SH, Yeo SG
Abstract · Sentences

**Bell's Palsy in Children (BellPIC): protocol for a multicentre, placebo-controlled randomized trial**
Babl FE, Mackay MT, Borland ML, Herd DW, Kochar A, Hort J, Rao A, Cheek JA, Furyk J, Barrow L, George S, Zhang...
Abstract

**Assessing recording delays in general practice records to inform near real-time vaccine safety surveillance using the Clinical Practice Research Datalink (CPRD)**
Leite A, Andrews NJ, Thomas SL
Abstract · Sentences · Table 1

*Figure 26. PubMiner result page showing a set of returned articles*



PubMiner Demographic-Enriched PubMed Search

**Acupuncture and vitamin B12 injection for Bell's palsy: no high-quality evidence exists**
Wang LL, Guan L, Hao PL, Du JL, Zhang MX
Abstract Sentences

**Comparison of the Efficacy of Combination Therapy of Prednisolone - Acyclovir with Prednisolone Alone in Bell's Palsy**
KHAJEH A, FAYYAZI A, SOLEIMANI G, MIRI-ALIABAD G, SHAYKH VEISI S, KHAJEH B
Abstract Sentences Table 1

Publication Date: **2015 Spring** PubMed: 26221158 PubMed Central: 4515336

Article Abstract

Demographic Sentences

Table 1

Table 1 Extract

Frequency of Bell's palsy Patients according to the Age and Gender in Two Groups

| Group | | Prednisolone group | Combined Group | P-Value |
|---|---|---|---|---|
| Gender | Male | 14(61 %) | 9(45%) | 0.29 |
| | Female | 9(39% ) | 11(55% ) | |
| Mean age (yr) | | 8.65±5.07 | 8.35±4.92 | 0.84 |

**Single dose intravenous methyl prednisolone versus oral prednisolone in Bell's palsy: A randomized controlled trial**
Giri P, Garg RK, Singh MK, Verma R, Malhotra HS, Sharma PK
Abstract Sentences

*Figure 27. PubMiner result page showing an expanded row*

## Final User Stories

## System Installation Manual

There are two general components to the PubMiner system: the web application and the data pipeline. The web application is stored in a github repository that can be cloned from https://github.com/CS599-MEDLINE/pubminer-web-app. Instructions for setting up and running the web application are found both in the README of the repository and the Developer Installation Manual section, below.

## Backend services on AWS: DynamoDB, S3 buckets, and AWS Lambda setup

The backend elements are running on AWS using the services AWS Lambda, DynamoDB, IAM, S3, EC2, Cloud Formation and CloudWatch. The installation and configuration of the DynamoDB tables and the S3 buckets has been performed with the boto3 library in Python 3.6. The IAM roles and policies, the AWS Lambda functions and the associated event triggers are created with a AWS Cloud Formation template. The entire backend can be installed from scratch using Python and boto3, first creating the database tables and S3 buckets (or using the ones in place, as "*pubmedcentral_oa*" is about 40 GB) and then launching the Cloud Formation template with boto3. Every step is provided with instructions in a Jupyter notebook setup.ipynb.

Currently all of the AWS services are running from a single account, and the AWS IAM service has been used to configure access for all team members, through a user Group and Role permissions. While the aspects of access management for our customer are out of scope for the project, a similar Group/Role approach can be taken after handover. The relevant sections of the Cloud Formation template would need adjusting for the Region and AccountID references.

### DynamoDB databases

The DynamoDB tables "*demographics*" and "*demographics_meta*"  are created with just the primary key elements, and throughput capacities. The remaining attributes are created populated by the Lambda functions. The tables are created in setup.ipynb using boto3. There are also a full set of maintenance functions in the Jupyter notebook create_table.ipynb (batch populate, inquiring about the tables, database queries, updating, deleting items, etc.).

### S3 buckets

The S3 bucket "*pubmedcentral_oa*" holds all of the XML files for the Open Access articles which are the sources of information in the database (these are .nxml files). The S3 bucket "*pubminer_upload*" contains the configuration files, such as the Cloud Formation template, the Sentence Miner application jar file, and the source packages for the AWS Lambda functions. Both of these S3 buckets are created using boto3 (in setup.ipynb).

### AWS Lambda functions and AWS Cloud Formation

The Lambda functions can be setup using the an AWS Cloud Formation template. The template, as well as the Lambda function packages are created using Python 3.6 in the Jupyter notebook setup.ipynb. The notebook is used to create (and adjust if necessary) the .py files for the Lambda functions and can then be used to upload them to S3. Similarly, the Cloud Formation template is created and uploaded to S3 from the notebook, and then the Cloud Formation stack is launched using boto3.

Each Lambda function is described in a dedicated section, along with the trigger events, and the example ARN for the running platform. A brief overview of the Lambda functions follows:

- "GetPMCUpdatesFromCSV" does the search, intersection of PMIDs and PMCIDs for the Open Access subset, and the initial population of items in the demographics and demographics_meta tables.
- "DownloadPMC-OAFileToS3" does the download of files from PMC-OA and the upload to S3.
- "TruncateTable" does the table truncation. The AWS Lambda package includes the BeautifulSoup4 and lxml packages, which are not included by default on Lambda.
- The sentence extraction is performed on an EC2 instance, which is launched with a "user-data" script to perform the installation of the packages and necessary .jar file to run. The file "SentenceMinerOnEC2Instance" does these steps.
- The function "UpdateStatsInDemographicsMeta" does a final update to the demographics_meta dynamodb table with the statistics of the latest update to the demographics table

## Roles and Policies

Three AWS Roles are created from within the Cloud Formation template, along with their associated AWS Policy documents. They are:

- The role "PM_lambda-dynamo-execution-role" which is for the Lambda functions which poll the dynamodb streams, and which write to the dynamodb tables and S3.
- The role "PM_lambda_start_stop_ec2" which is for the sentence extraction Lambda function to poll the dynamodb stream, launch the EC2 instance, as well as write to the dynamodb tables and S3.
- The role "PM_lambda_dynamodb_S3_role" is for the other Lambda functions which do not require permission to invoke functions or launch instances, but require permissions to write to Dynamodb or to Put to or Get from S3 buckets.

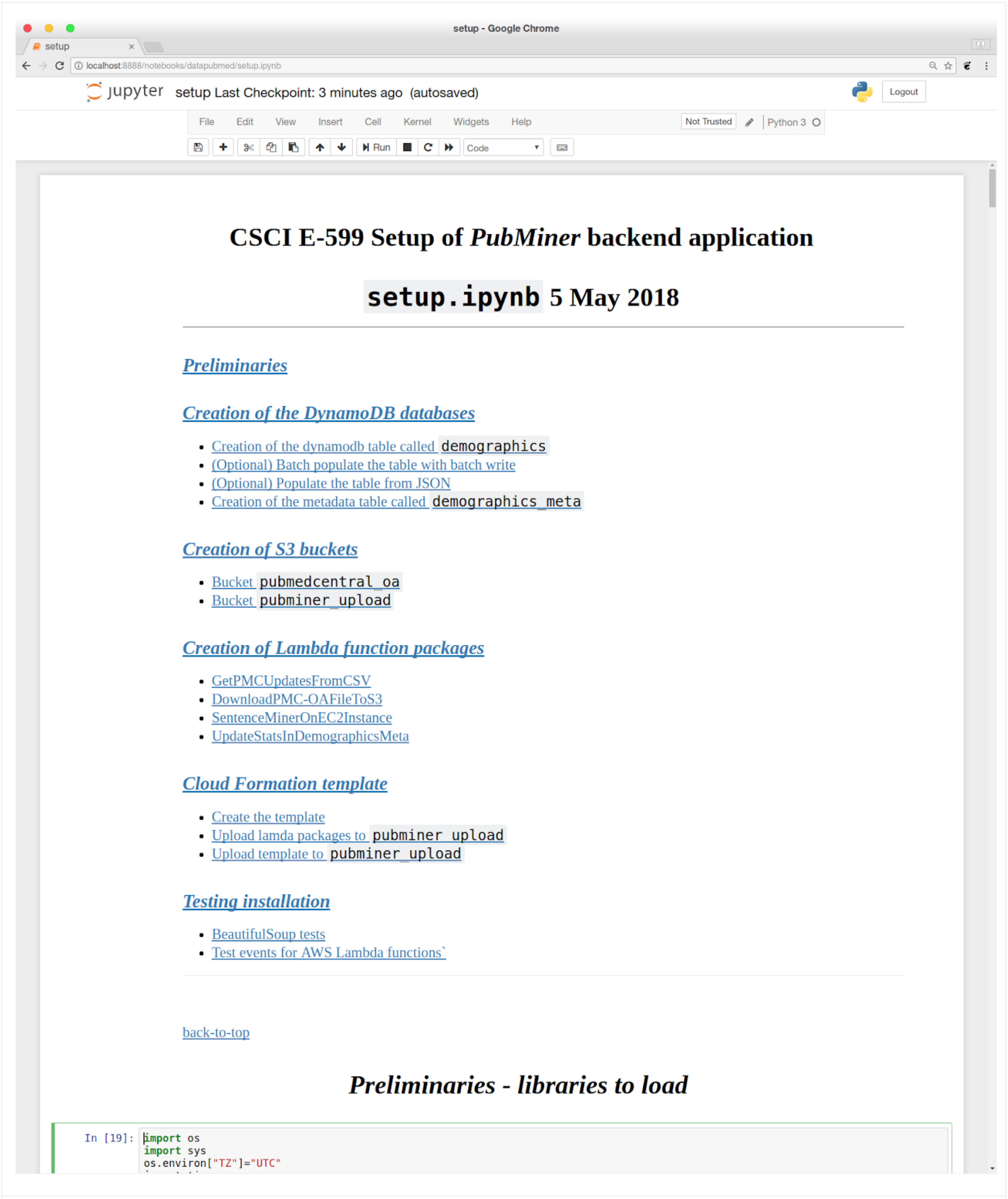
The Policy documents are included in the Role creation template, as inline Policies.

## Testing the installation

The file setup.ipynb contains the JSON code for the test events of the individual Lambda functions. In addition, the set of BeautifulSoup tests can be run from the notebook to confirm that the table mining functions are working as expected.

## Note for Full Installation

It should be noted that if the entire backend needs to be installed from scratch, the demographics table can be populated, if necessary, for the entire set of PMCIDs, using the Python function (since an AWS Lambda function GrabUpdatesPMCfromCSV.py of such magnitude would time out). Once the primary key items for "pmcid" are populated, the rest of the backend will automatically generate from the sequence of Lambda functions, triggered initially from the stream for the demographics dynamodb table.



CSCI E-599 Setup of *PubMiner* backend application

`setup.ipynb` 5 May 2018

***Preliminaries***

***Creation of the DynamoDB databases***

- Creation of the dynamodb table called `demographics`
- (Optional) Batch populate the table with batch write
- (Optional) Populate the table from JSON
- Creation of the metadata table called `demographics_meta`

***Creation of S3 buckets***

- Bucket `pubmedcentral_oa`
- Bucket `pubminer_upload`

***Creation of Lambda function packages***

- GetPMCUpdatesFromCSV
- DownloadPMC-OAFileToS3
- SentenceMinerOnEC2Instance
- UpdateStatsInDemographicsMeta

***Cloud Formation template***

- Create the template
- Upload lamda packages to `pubminer_upload`
- Upload template to `pubminer_upload`

***Testing installation***

- BeautifulSoup tests
- Test events for AWS Lambda functions`

back-to-top

***Preliminaries - libraries to load***

```
In [19]: import os
         import sys
         os.environ["TZ"]="UTC"
```

*Figure 28. Setup of backend application in AWS - setup.ipynb*

## Developer Installation Manual

The following is a guide to building and deploying the web application.

### Running PubMiner Locally

#### Install Required Software and Tools

- Install Node.js® and npm (node package manager). The installer at nodejs.org will handle both of those for you
- Install Bower globally `npm install -g bower`
- Install the gulp command line tools globally `npm install -g gulp-cli`

#### Install and Build Application Dependencies

- Clone the source code repository from https://github.com/CS599-MEDLINE/pubminer-web-app: `git clone <url> <project directory name>`
- Ensure you are in the project directory
- Run `npm install` to install the server-side dependencies (express, pug, xml-simple, etc.)
- Run `bower install` to install client-side dependencies (PatternFly, jQuery)
- Run `gulp` to build pmstyles.css and automatically have it copied to the public/css folder

#### Start the Application

- Ensure you are in the project directory
- Run `npm start` to start the server on port 8081
    - If you'd like to change the port to something different, you can modify the bin\www file
    - To start in development mode, run `npm run start-dev` instead. In development mode, source changes are detected and the server automatically restarts. Also, the browser developer tools will display a Node.js icon that will allow you to debug server side code
- In your browser, navigate to http://localhost:8081

### Running PubMiner in the Cloud

#### Setup

- Create a new virtual machine for the application on the cloud provider of choice (e.g. AWS, Digital Ocean). Details vary by provider.

- Connect to a terminal window for the virtual machine.
- Follow the steps "Required Software and Tools" and "Install and Build Application Dependencies" above.
- Install Forever globally `npm install -g forever`. Forever will start an application and automatically restart an application that has crashed.

### Start the Application

- Connect to a terminal window for the virtual machine.
- Start the application using Forever: `forever start ./bin/www`
- View the details using: forever list. This will show the uid, log file location, and uptime.

### Update the Running Code

- Connect to a terminal window for the virtual machine.
- Find the uid assigned to the application: forever list
- Stop the application: `forever stop <uid>`
- Switch to the project directory
- Refresh the source code from GitHub: `git pull origin master`
- Install any new dependencies: `npm install`
- Generate the .css: `gulp`
- Start the application: `forever start ./bin/www`

## Additional Information

### Running Integration Tests with Cypress

- Ensure the application and database are running
- Run `npm run cypress:open`
- Select search_spec.js or home spec.js from the Integration Test list. The tests should automatically start executing

### Running Unit Tests with Mocha

- Ensure the application and database are running
- Run `npm run test`
- Results are logged to the console

### PubMed API key

The app will recognize a PubMed API key stored as configuration value. The PubMed API key should be put in local.json on the deployed machine under the config folder. The example below shows how to provide the API key for the application to use in service calls.

```
<root>/config/local.json
```

```
{
  "PubMedService": {
    "queryOptions": {
      "api_key": "<your-api-key-here>"
    }
  }
}
```

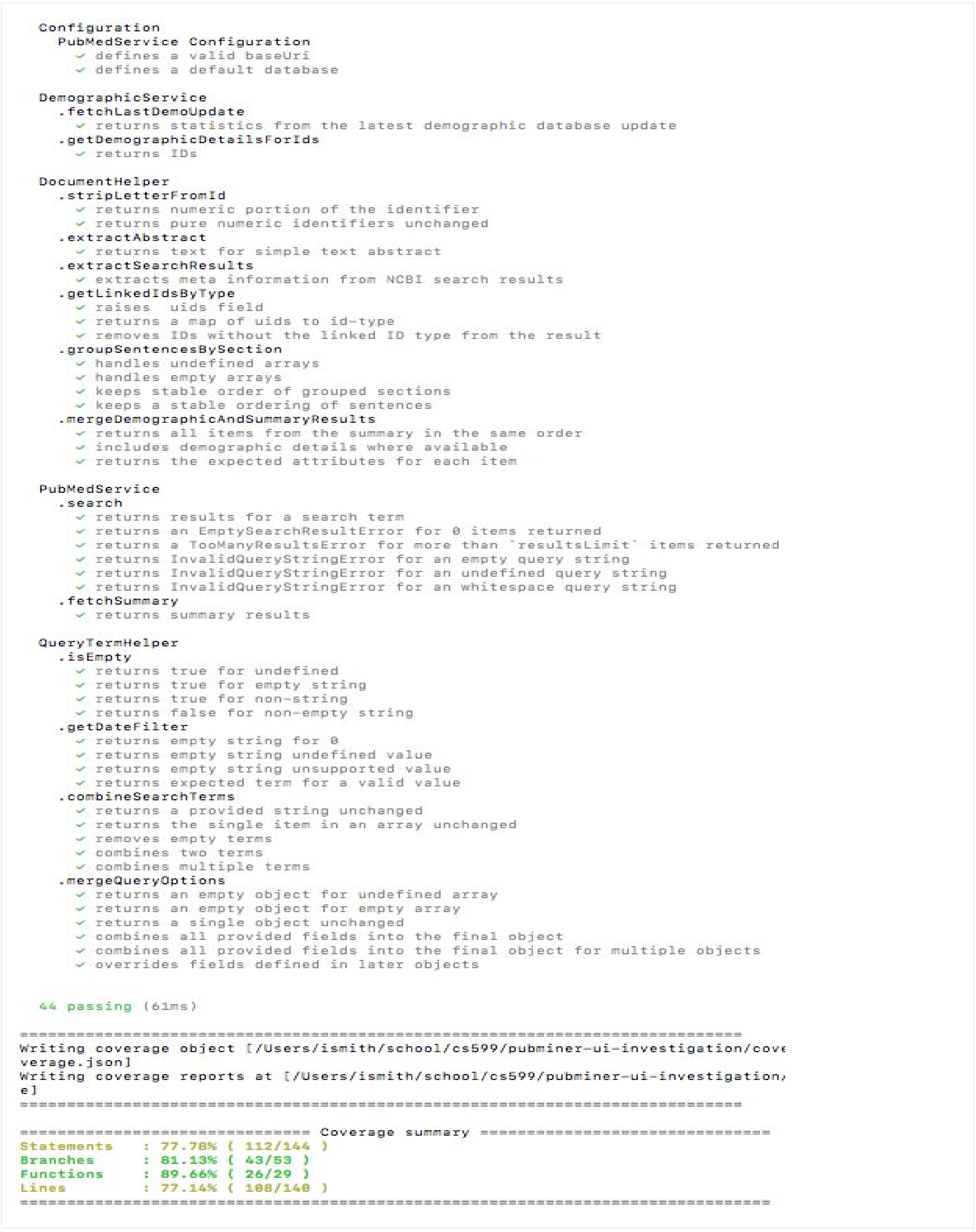
The project's .gitignore file is configured to not permit check-in of local.json.

### Building pmstyles.css

PatternFly uses Less to compile PatternFly and Bootstrap style elements and project-specific styles into a single .css file. The project file less\pmstyles.less describes how the target css file is assembled, and is the place to add additional PubMiner-specific styles.

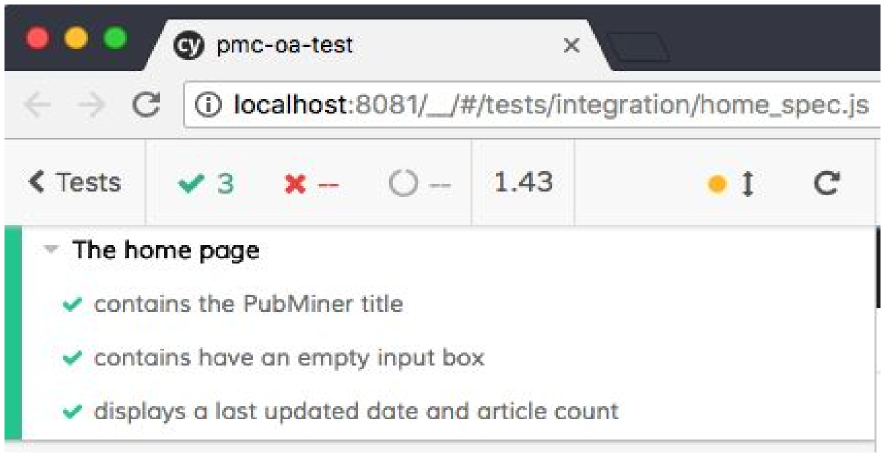
The project uses gulp to compile the Less script. To build pmstyles.css, install the gulp command line tools globally `npm install -g gulp-cli`. You may also need to run `npm install` within the project folder to update the project-specific dependencies (like gulp, gulp-less and gulp-plumber). Once the dependencies are installed, you simply run `gulp` in the project folder to build pmstyles.css and automatically copy it to the \public\css folder.

## Automated Test Reports



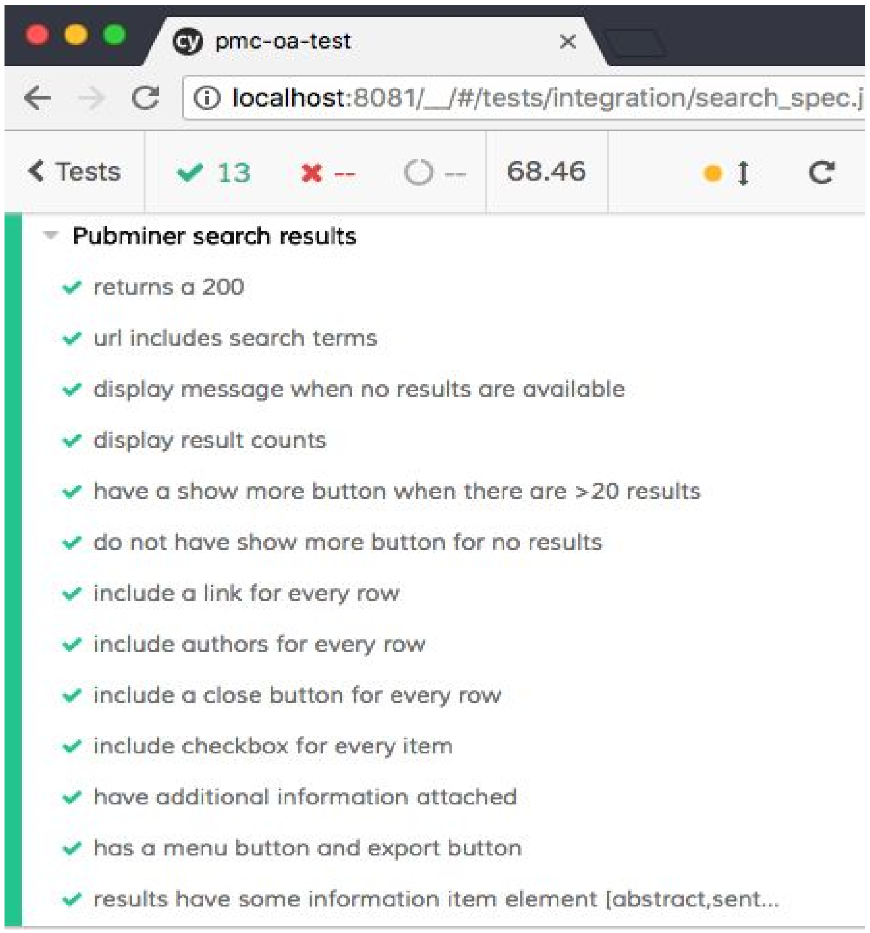
```
  Configuration
    PubMedService Configuration
      ✓ defines a valid baseUri
      ✓ defines a default database

  DemographicService
    .fetchLastDemoUpdate
      ✓ returns statistics from the latest demographic database update
    .getDemographicDetailsForIds
      ✓ returns IDs

  DocumentHelper
    .stripLetterFromId
      ✓ returns numeric portion of the identifier
      ✓ returns pure numeric identifiers unchanged
    .extractAbstract
      ✓ returns text for simple text abstract
    .extractSearchResults
      ✓ extracts meta information from NCBI search results
    .getLinkedIdsByType
      ✓ raises  uids field
      ✓ returns a map of uids to id-type
      ✓ removes IDs without the linked ID type from the result
    .groupSentencesBySection
      ✓ handles undefined arrays
      ✓ handles empty arrays
      ✓ keeps stable order of grouped sections
      ✓ keeps a stable ordering of sentences
    .mergeDemographicAndSummaryResults
      ✓ returns all items from the summary in the same order
      ✓ includes demographic details where available
      ✓ returns the expected attributes for each item

  PubMedService
    .search
      ✓ returns results for a search term
      ✓ returns an EmptySearchResultError for 0 items returned
      ✓ returns a TooManyResultsError for more than `resultsLimit` items returned
      ✓ returns InvalidQueryStringError for an empty query string
      ✓ returns InvalidQueryStringError for an undefined query string
      ✓ returns InvalidQueryStringError for an whitespace query string
    .fetchSummary
      ✓ returns summary results

  QueryTermHelper
    .isEmpty
      ✓ returns true for undefined
      ✓ returns true for empty string
      ✓ returns true for non-string
      ✓ returns false for non-empty string
    .getDateFilter
      ✓ returns empty string for 0
      ✓ returns empty string undefined value
      ✓ returns empty string unsupported value
      ✓ returns expected term for a valid value
    .combineSearchTerms
      ✓ returns a provided string unchanged
      ✓ returns the single item in an array unchanged
      ✓ removes empty terms
      ✓ combines two terms
      ✓ combines multiple terms
    .mergeQueryOptions
      ✓ returns an empty object for undefined array
      ✓ returns an empty object for empty array
      ✓ returns a single object unchanged
      ✓ combines all provided fields into the final object
      ✓ combines all provided fields into the final object for multiple objects
      ✓ overrides fields defined in later objects


  44 passing (61ms)

=============================================================================
Writing coverage object [/Users/ismith/school/cs599/pubminer-ui-investigation/cove
verage.json]
Writing coverage reports at [/Users/ismith/school/cs599/pubminer-ui-investigation/
e]
=============================================================================

=============================== Coverage summary ===============================
Statements   : 77.78% ( 112/144 )
Branches     : 81.13% ( 43/53 )
Functions    : 89.66% ( 26/29 )
Lines        : 77.14% ( 108/140 )
================================================================================
```

*Figure 29. Mocha Unit Test Results*



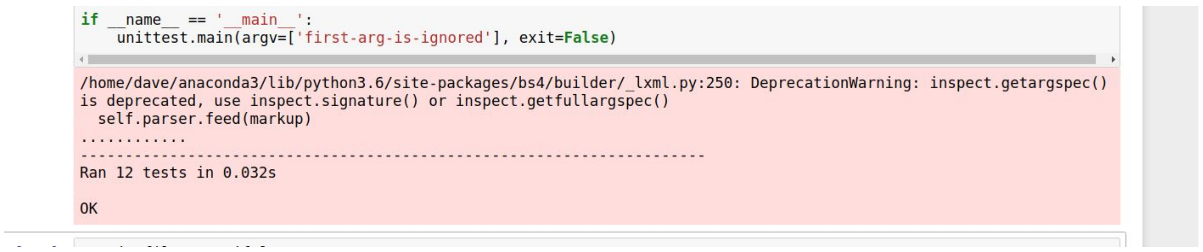
The home page
- ✓ contains the PubMiner title
- ✓ contains have an empty input box
- ✓ displays a last updated date and article count

*Figure 30a. Cypress Integration Test Results - Home Page*



Pubminer search results
- ✓ returns a 200
- ✓ url includes search terms
- ✓ display message when no results are available
- ✓ display result counts
- ✓ have a show more button when there are >20 results
- ✓ do not have show more button for no results
- ✓ include a link for every row
- ✓ include authors for every row
- ✓ include a close button for every row
- ✓ include checkbox for every item
- ✓ have additional information attached
- ✓ has a menu button and export button
- ✓ results have some information item element [abstract,sent...

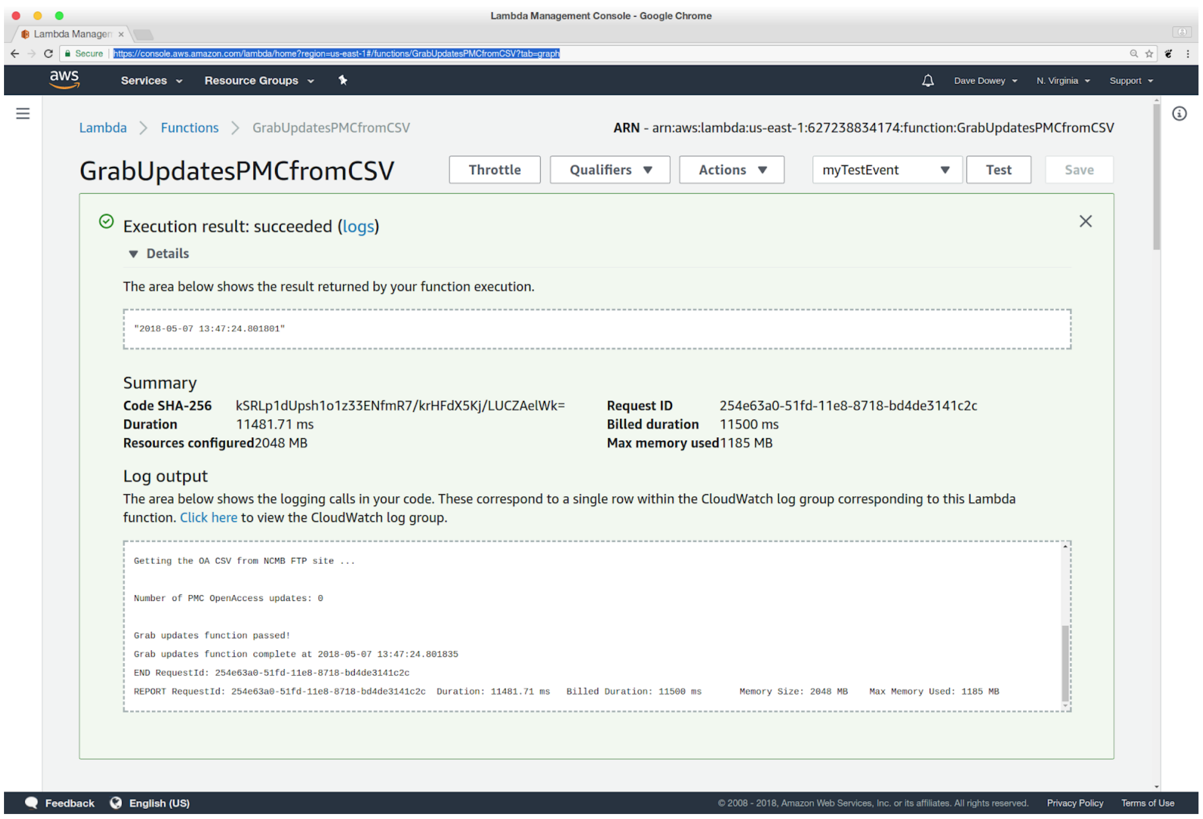
*Figure 30b. Cypress Integration Test Results - Search Results Page*



```
if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
/home/dave/anaconda3/lib/python3.6/site-packages/bs4/builder/_lxml.py:250: DeprecationWarning: inspect.getargspec()
is deprecated, use inspect.signature() or inspect.getfullargspec()
  self.parser.feed(markup)
............
----------------------------------------------------------------------
Ran 12 tests in 0.032s

OK
```

*Figure 31. Results from BeautifulSoup tests (from crummy.com) of Table Mining*



Lambda Management Console - Google Chrome
https://console.aws.amazon.com/lambda/home?region=us-east-1#/functions/GrabUpdatesPMCfromCSV?tab=graph
Services
Resource Groups
Dave Dowey
N. Virginia
Support
Lambda > Functions > GrabUpdatesPMCfromCSV
Throttle
Qualifiers
Actions
myTestEvent
Test
Save
Feedback
English (US)
Privacy Policy
Terms of Use

ARN - arn:aws:lambda:us-east-1:627238834174:function:GrabUpdatesPMCfromCSV

GrabUpdatesPMCfromCSV

Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution.

```
"2018-05-07 13:47:24.801801"
```

Summary

| | | | |
|---|---|---|---|
| **Code SHA-256** | kSRLp1dUpsh1o1z33ENfmR7/krHFdX5Kj/LUCZAelWk= | **Request ID** | 254e63a0-51fd-11e8-8718-bd4de3141c2c |
| **Duration** | 11481.71 ms | **Billed duration** | 11500 ms |
| **Resources configured** | 2048 MB | **Max memory used** | 1185 MB |

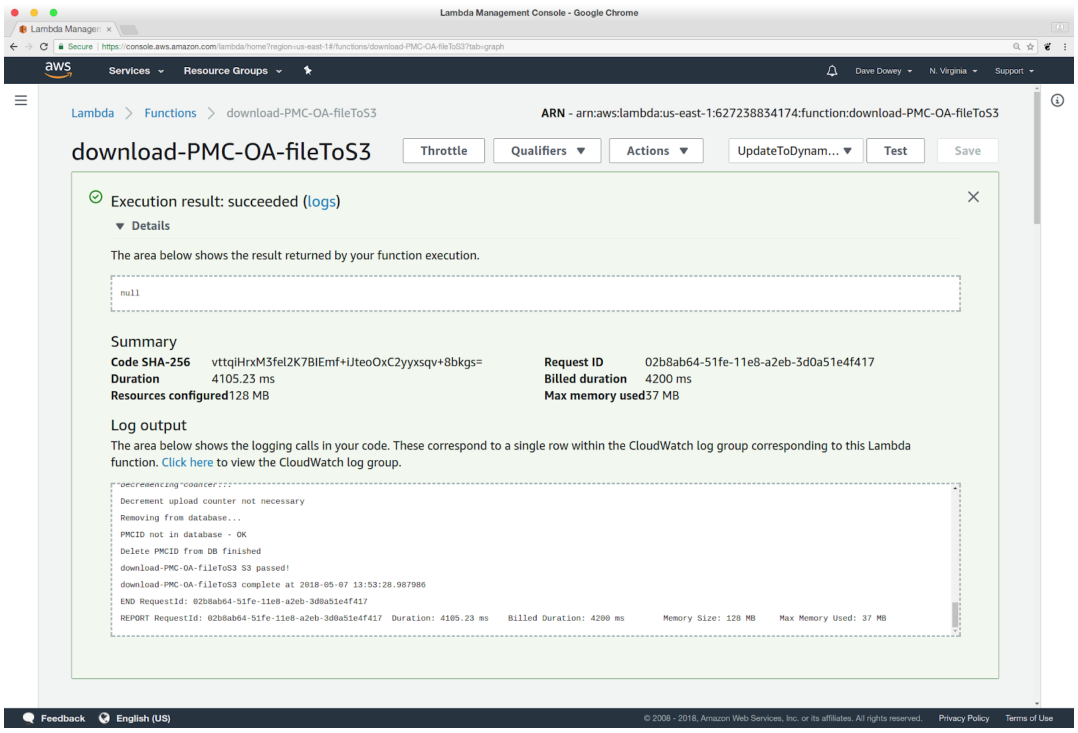
Log output

The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.

```
Getting the OA CSV from NCMB FTP site ...


Number of PMC OpenAccess updates: 0


Grab updates function passed!
Grab updates function complete at 2018-05-07 13:47:24.801835
END RequestId: 254e63a0-51fd-11e8-8718-bd4de3141c2c
REPORT RequestId: 254e63a0-51fd-11e8-8718-bd4de3141c2c  Duration: 11481.71 ms   Billed Duration: 11500 ms       Memory Size: 2048 MB    Max Memory Used: 1185 MB
```

© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.

*Figure 32a. Test of AWS Lambda function to Get Updates from PMC with cron-job trigger*


Lambda Management Console - Google Chrome
https://console.aws.amazon.com/lambda/home?region=us-east-1#/functions/download-PMC-OA-fileToS3?tab=graph
Services
Resource Groups
Dave Dowey
N. Virginia
Support
Lambda > Functions > download-PMC-OA-fileToS3
Throttle
Qualifiers
Actions
UpdateToDynam...
Test
Save
Feedback
English (US)
© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

**ARN** - arn:aws:lambda:us-east-1:627238834174:function:download-PMC-OA-fileToS3

download-PMC-OA-fileToS3

Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution.

```
null
```

Summary

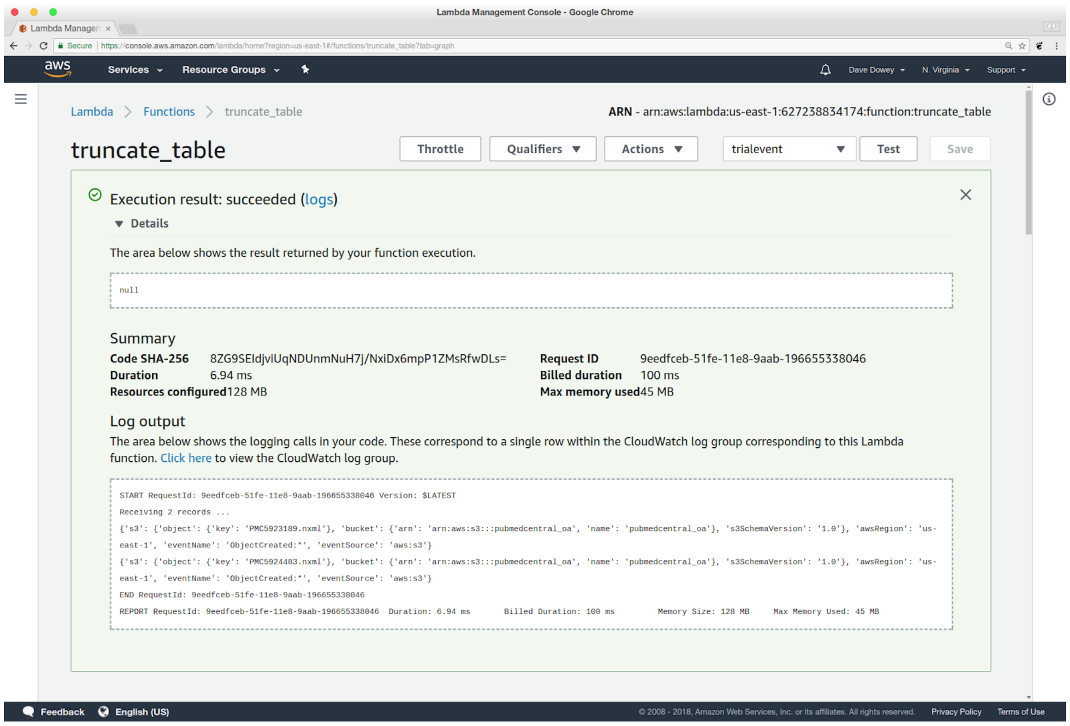
| | | | |
|---|---|---|---|
| **Code SHA-256** | vttqiHrxM3fel2K7BIEmf+iJteoOxC2yyxsqv+8bkgs= | **Request ID** | 02b8ab64-51fe-11e8-a2eb-3d0a51e4f417 |
| **Duration** | 4105.23 ms | **Billed duration** | 4200 ms |
| **Resources configured** | 128 MB | **Max memory used** | 37 MB |

Log output

The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.

```
Decrementing counter...
Decrement upload counter not necessary
Removing from database...
PMCID not in database - OK
Delete PMCID from DB finished
download-PMC-OA-fileToS3 S3 passed!
download-PMC-OA-fileToS3 complete at 2018-05-07 13:53:28.987986
END RequestId: 02b8ab64-51fe-11e8-a2eb-3d0a51e4f417
REPORT RequestId: 02b8ab64-51fe-11e8-a2eb-3d0a51e4f417  Duration: 4105.23 ms    Billed Duration: 4200 ms        Memory Size: 128 MB     Max Memory Used: 37 MB
```

*Figure 32b. Test of AWS Lambda function for putting OA file to S3 with a DynamoDB trigger*


Lambda Management Console - Google Chrome
https://console.aws.amazon.com/lambda/home?region=us-east-1#/functions/truncate_table?tab=graph
Services
Resource Groups
Dave Dowey
N. Virginia
Support
Lambda > Functions > truncate_table
Throttle
Qualifiers
Actions
trialevent
Test
Save
Feedback
English (US)
© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

**ARN** - arn:aws:lambda:us-east-1:627238834174:function:truncate_table

truncate_table

Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution.

```
null
```

Summary

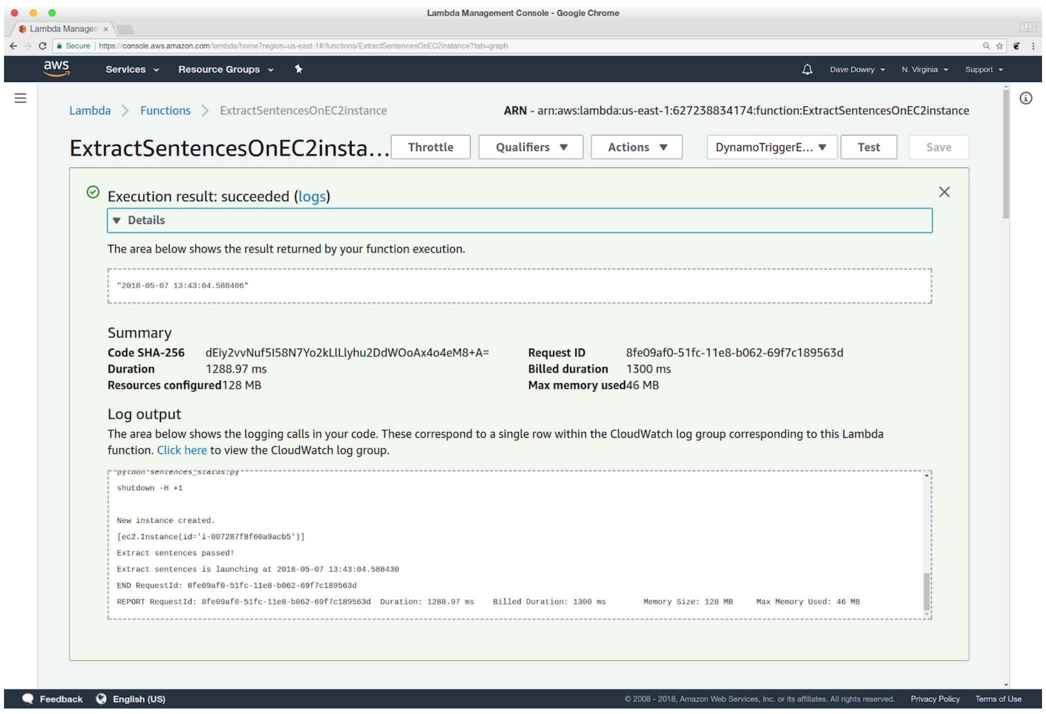
| | | | |
|---|---|---|---|
| **Code SHA-256** | 8ZG9SEIdjviUqNDUnmNuH7j/NxiDx6mpP1ZMsRfwDLs= | **Request ID** | 9eedfceb-51fe-11e8-9aab-196655338046 |
| **Duration** | 6.94 ms | **Billed duration** | 100 ms |
| **Resources configured** | 128 MB | **Max memory used** | 45 MB |

Log output

The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.

```
START RequestId: 9eedfceb-51fe-11e8-9aab-196655338046 Version: $LATEST
Receiving 2 records ...
{'s3': {'object': {'key': 'PMC5923189.nxml'}, 'bucket': {'arn': 'arn:aws:s3:::pubmedcentral_oa', 'name': 'pubmedcentral_oa'}, 's3SchemaVersion': '1.0'}, 'awsRegion': 'us-
east-1', 'eventName': 'ObjectCreated:*', 'eventSource': 'aws:s3'}
{'s3': {'object': {'key': 'PMC5924483.nxml'}, 'bucket': {'arn': 'arn:aws:s3:::pubmedcentral_oa', 'name': 'pubmedcentral_oa'}, 's3SchemaVersion': '1.0'}, 'awsRegion': 'us-
east-1', 'eventName': 'ObjectCreated:*', 'eventSource': 'aws:s3'}
END RequestId: 9eedfceb-51fe-11e8-9aab-196655338046
REPORT RequestId: 9eedfceb-51fe-11e8-9aab-196655338046  Duration: 6.94 ms       Billed Duration: 100 ms         Memory Size: 128 MB     Max Memory Used: 45 MB
```

*Figure 32c. Test of AWS Lambda function for Table Mining with S3 file upload trigger*


Lambda Management Console - Google Chrome
Lambda Managem
Secure | https://console.aws.amazon.com/lambda/home?region=us-east-1#/functions/ExtractSentencesOnEC2instance?tab=graph
aws
Services
Resource Groups
Dave Dowey
N. Virginia
Support
Lambda > Functions > ExtractSentencesOnEC2instance
Throttle
Qualifiers
Actions
DynamoTriggerE...
Test
Save
Details
Feedback
English (US)
© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

**ARN** - arn:aws:lambda:us-east-1:627238834174:function:ExtractSentencesOnEC2instance

ExtractSentencesOnEC2insta...

Execution result: succeeded (logs)

The area below shows the result returned by your function execution.

```
"2018-05-07 13:43:04.588406"
```

Summary

| | | | |
|---|---|---|---|
| **Code SHA-256** | dEiy2vvNuf5I58N7Yo2kLlLlyhu2DdWOoAx4o4eM8+A= | **Request ID** | 8fe09af0-51fc-11e8-b062-69f7c189563d |
| **Duration** | 1288.97 ms | **Billed duration** | 1300 ms |
| **Resources configured** | 128 MB | **Max memory used** | 46 MB |

Log output

The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.

```
python sentences_status.py
shutdown -H +1


New instance created.
[ec2.Instance(id='i-007287f8f60a9acb5')]
Extract sentences passed!
Extract sentences is launching at 2018-05-07 13:43:04.588430
END RequestId: 8fe09af0-51fc-11e8-b062-69f7c189563d
REPORT RequestId: 8fe09af0-51fc-11e8-b062-69f7c189563d  Duration: 1288.97 ms    Billed Duration: 1300 ms        Memory Size: 128 MB     Max Memory Used: 46 MB
```

*Figure 32d. Test of AWS Lambda function for Sentence Miner with a DynamoDB trigger*