Fall 2015

GRAPH BASESD WORD SENSE DISAMBIGUATION FOR CLINICAL ABBREVIATIONS USING APACHE SPARK

Veebha Padavkar
San Jose State University

GRAPH BASESD WORD SENSE DISAMBIGUATION FOR CLINICAL ABBREVIATIONS USING APACHE SPARK

A Thesis

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirement of the Degree

Master of Science

By

Veebha Padavkar

Dec 2015

The Designated Thesis Committee Approves the Thesis Titled

GRAPH BASESD WORD SENSE DISAMBIGUATION FOR CLINICAL ABBREVIATIONS USING APACHE SPARK

by

Veebha Padavkar

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

DEC 2015

ABSTRACT

GRAPH BASESD WORD SENSE DISAMBIGUATION FOR CLINICAL ABBREVIATIONS USING APACHE SPARK

Identification of the correct sense for an ambiguous word is one of the major challenges for language processing in all domains. Word Sense Disambiguation is the task of identifying the correct sense of an ambiguous word by referencing the surrounding context of the word. Similar to the narrative documents, clinical documents suffer from ambiguity issues that impact automatic extraction of correct sense from the document. In this project, we propose a graph-based solution based on an algorithm originally implemented by Osmar R. Zaine et al. for word sense disambiguation specifically focusing on clinical text. The algorithm makes use of proposed UMLS Metathesaurus as its source of knowledge. As an enhancement to the existing implementation of the algorithm, this project uses Apache Spark - A Big Data Technology for cluster based distributed processing and performance optimization.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Thanh Tran, my project advisor, for his amazing support and motivation throughout this project.

I would like to thank my project committee member, Professor Thomas Austin and Chandni Patel for their contribution in completion of this project

I would like to thank my family for their great support throughout my studies here at San Jose State University

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

The exponential growth of data and data processing tools in the 21st century has brought an enormous amount of information growth in a brief span of time. Data is gathered from numerous viewpoints that generate large volumes of raw information with exceptionally varied characteristics. Simple quantification can be inferred by understanding that Information Organizations today are processing petabytes of information every day with an even higher velocity of data generation by users, sensors and mobile devices. Considering the possibility of extracting essentially important insights from the data, processing and analytics has become continued processes for decades now

One such domain that considers knowledge extraction as the prime benefit of the data for the betterment of future is the Bio-medical domain. Tremendous volumes of data with exceptionally high velocity of data generation and varied data characteristics make this domain a unique candidate for solving numerous data processing problems. The notable feature of data in this domain is the existence of domain specific terms that essentially requires correct understanding of the data as a whole to infer the meaning of any part of the data. As the data grows, there is possibility of increased incorrect inference for the same part of data for which the inference might have been accurate in past. This essentially leads to core problem of lexical disambiguation that gets introduced as the volume of data increase and reduce the uniqueness of features in the data.

Humans usually distinguish the data and its usage from context. In a similar way, for machines to understand data, the context of the data or the words play an important role. Various systems have been implemented with advanced algorithms to learn the correct meaning of the data from its context. However, the unstructured nature of the data and irrelevant existence of context has always been a challenge for these systems.

Algorithms have been developed that try to interpret the correct meaning of the words from its surrounding context. However, the existence of ambiguity has such aspects that inferring the correct meaning are not merely possibly by understanding the surrounding words. It has become critical to understand ‘sense’ of the complete data surrounding an ambiguous content. Approaches to automatically disambiguating words therefore typically make use of context words to learn the sense of the data as a whole. This problem has been discussed and worked on by many experts of general-purpose language as well as domain specific language experts and gave rise to the problem domain as Word Sense Disambiguation (WSD) which essentially focus on inferring the correct meaning of the words through the sense of surrounded context.

WSD techniques have evolved over the period by adopting la test tools and technologies. However, the most essential barrier in improved disambiguation results is the lack of formally annotated data that can directly help to quickly infer the correct meanings of ambiguous word instances. Hence, WSD systems essentially operate on subset of clinical data on which the accuracy of their system is highly dependent.

Annotation of text detected in clinical settings, such as nurses and discharge summaries notes is particularly expensive in time and resources since it has to be performed by medical experts. Hence many efforts in WSD required either unsupervised or semi-supervised methods that resulted in little to almost no data being annotated. The knowledge base being specific to domains are proving helpful in improving the results of systems that were previously less efficient with the need of good and correct annotated data. Hence, the algorithms moved towards the use of complete knowledge base for disambiguation of the terms with unsupervised methods. In this project we have implemented one such WSD system, a graph based unsupervised approach, built on Apache Spark.

This project offers a solution to the problem of disambiguation by utilizing the bio-medical domain specific knowledge base on clinical text with no cost required to manually annotate the data as we use an unsupervised approach. We have considered using and examining results on Apache Spark, which is an interesting big data topic that has a lot of developing potential. This project extends one aspect of the graph based WSD system by optimizing the developed algorithm and comparing results when executed on Apache spark by calibrating the performance.

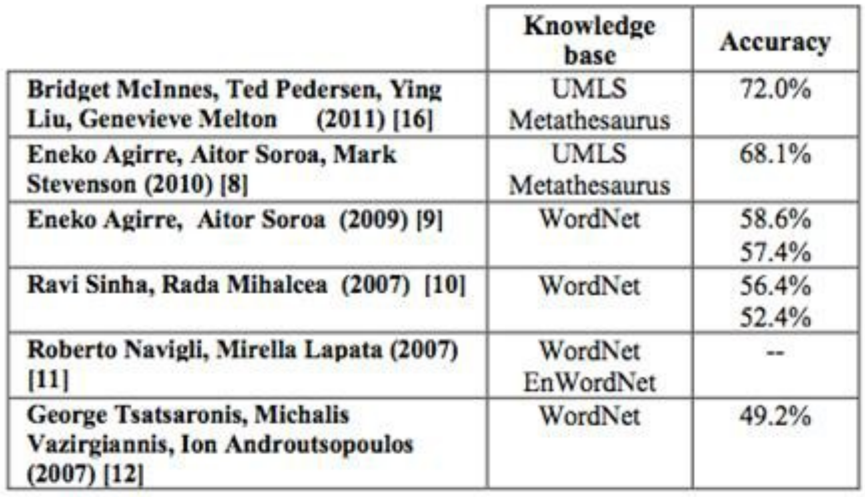
This project introduces an implementation of unsupervised graph based approach in the biomedical domain, which uses the unified Medical Language System as Knowledge Base. Several tactics on how efficiently the data is stored in Sparks Resilient Distributed Datasets to leverage the in-memory processing capabilities.

The performance of executing and solving the disambiguation heavily depends on the intrinsic optimization of the algorithm itself and the technologies that make up the systems on which it is implemented. Hence, this project aims at using a state of the art setup to implement a proven algorithm with enhancements focusing on improved accuracy and performance optimization. The system is evaluated with practical datasets; large enough to simulate how professional WSD system would work in a minimized scale. Several metrics are tested to compare performances of the chosen strategies and scoring schemes.

This report categorically discusses the graph based unsupervised word sense disambiguation. Chapter 2 discusses on related work on WSD system and resources required by our system. Chapter 3 discusses our unsupervised graph based approach to WSD using the UMLS Metathesaurus. Chapter 4 presents the evaluation of our algorithm. Chapter 5 concludes our findings with results.

2 RELATED WORKS AND BACKGROUND

WSD Systems are disambiguation systems that are not dependent on any domain, which means that these systems are not customized for any particular field or domain. Knowledge base is the main key part in any unsupervised WSD system, since the entire disambiguation process is dependent on the knowledgebase. For instance the UMLS (Unified Medical Language System) knowledge base is normally utilized by WSD systems, which concentrate on the biomedical area while WordNet is regularly utilized by area, free WSD. In Table I we present six late unsupervised diagram based WSD calculations alongside their insight base, and the reported precision. As the reported exactness appears, biomedical WSD accomplish better precision contrasted with their space autonomous partner. [6]



	Knowledge base	Accuracy
Bridget McInnes, Ted Pedersen, Ying Liu, Genevieve Melton (2011) [16]	UMLS Metathesaurus	72.0%
Eneko Agirre, Aitor Soroa, Mark Stevenson (2010) [8]	UMLS Metathesaurus	68.1%
Eneko Agirre, Aitor Soroa (2009) [9]	WordNet	58.6% 57.4%
Ravi Sinha, Rada Mihalcea (2007) [10]	WordNet	56.4% 52.4%
Roberto Navigli, Mirella Lapata (2007) [11]	WordNet EnWordNet	--
George Tsatsaronis, Michalis Vazirgiannis, Ion Androutsopoulos (2007) [12]	WordNet	49.2%

Figure 1 Unsupervised Graph based WSD Approaches

Since in our methodology we use UMLS as our knowledgebase and simulate Metamap as our concept mapping approach, we would further discuss these two in the following section.

2.1 Unified Medical Language System

The U.S. National Library of Medicine (NLM) has created an archival of different biomedical and clinical examination vocabularies to enhance the biomedical domain called as UMLS. UMLS is made out of the following three information sources:

The Metathesaurus is a vocabulary database of biomedical concepts with their diverse names, and connections between them. The Metathesaurus of the UMLS 2015AB contains more than 2.7 million concepts gathered from 161 vocabularies, for example, SNOMED Clinical Terms (SNOMED-CT) and Medical Subject Headings (MSH). The Metathesaurus sorts out knowledge taking into account concepts, where a Concept Unique Identifier (CUI) distinguishes every concept. [6]

The Semantic network, is a collection of semantic types which helps in categorizing all concepts which are represented in the metathesaurus, and also a collection of different semantic relations which defines possible relationships between different semantic types. The semantic network in the UMLS 2014AB contains:

- Contains 133 semantic types. Examples include, Enzyme, Genetic Function, Therapeutic, Laboratory procedure
- 54 Semantic Relations, examples include: affects, treats, disrupts, prevents. [6]

2.2 Metamap

Metamap is a concept mapping approach which was developed by the NLM to map biomedical texts to concepts in UMLS. It consists of five components.

- Lexical/Syntactic Analysis: This component segments biomedical text into phrases and later into terms. The text is Xerox part-of-speech tagged using the Xerox POS tagger. [6]
- Variant Generation: This segment creates a variation for every expression distinguished by the Lexical/Syntactic Analysis part. A variation is one or more expression words went with its spelling variations, derivational variations. [6]
- Candidate Identification: This segment recovers the arrangement of ideas from the UMLS Metathesaurus that contain no less than one variation recognized by the Variant Generation segment. [6]
- Candidate Evaluation: This part assesses every competitor against the data content. The mapping score is registered utilizing a blend of four linguistic measures: centrality; variety; scope; and cohesiveness. The four measures are consolidated straightly such that scope and cohesiveness get double the heaviness of centrality and variety. The score is standardized to a worth somewhere around 0 and 1,000, where a score of 1,000 means an immaculate applicant. [6]
- Mapping Construction: It integrates all the Metathesaurus candidates, which match the input text. [6]

2.3 Approaches for WSD

There are majorly two approaches for Word Sense Disambiguation, which are supervised Learning Approach and Unsupervised Learning Approach. Knowledgebase approach is also another approach which has been implemented by few older systems. Below are the description given for these approaches.

2.3.1 Knowledge Based Approach

Knowledge based approach for WSD involves use of dictionaries, thesaurus, ontologism, etc. to understand the sense of words in context. Even though these methods have comparatively lower performance than other approaches, but one advantage of this approach is that they do have large-scale knowledge resources.

However, few techniques are also using Knowledge Base approach. Since Knowledge Base Approaches tend to use external dictionary sources like WordNet etc. or some other machine language dictionary. Initial knowledge base approaches to WSD were dated back to the 1980's when experiments were piloted on very small domains. But the lack of large-scale computational assets did not allow a proper evaluation and comparison in end-to-end applications. To perform disambiguation process in knowledge base approach, hard coded grammatical rules are been used.

2.3.2 Supervised Learning Approach

Supervised learning method is that method which tries to find relationships between independent variables also known as input attributes and a target attribute also known as dependent variable. It makes use of labeled training data to derive functions. These derived functions are used further to predict results. This method is majorly implemented in different domains such as health, marketing, finance and manufacturing. The relationship found is represented in a structure referred to as a model. Usually models describe and explain phenomena, which are hidden in the dataset and can be used for predicting value of the target attribute knowing the values of the independent variables.

However, in supervised approaches, use of training data is involved. Generating training data manually requires lot of manual efforts plus the data can't be trusted on its accuracy. Since the training data does not have the inputs classified correctly, this can result in getting wrong disambiguated results. Hence due to the given reasons, unsupervised approach methods are considered more correct and accurate.

2.3.3 Unsupervised Learning Approach

In Unsupervised learning method it tries to find hidden structure in unlabeled data. Unsupervised methods for WSD can be broadly divided into two categories namely similarity-based and graph based ones. For graph based methods generally unsupervised approach is preferred since it offers an advantage of not requiring the training data.

2.3.3.1 Unsupervised WSD methods

Graph Based Method

Graph Based algorithms essentially consists of two stages. Initially, a graph is built considering all possible interpretations of the group of words from the knowledge base. The graph nodes represent the word senses, whereas edges represent the relationships between two nodes. In next step, the graph structure is examined to resolve importance of each node. Here sense disambiguation resolves to find the most important node for each word. The sense is primarily being disambiguated by traversing the graph and collecting the directly connected nodes for the word being disambiguated and applying a similarity metrics over the collected nodes.

Similarity Based Method

Similarity based algorithms assign a plausible sense to an ambiguous word by comparing each of its senses with those of the words surrounding the words to be disambiguated which are also referred to as the context words. The sense whose definition has the highest similarity is assumed to be the correct one. [4]

The algorithm calculations contrast in the similarity measure the y utilize and the received meaning of connection that can fluctuate from a couple of words to the entire corpus. In similarity based algorithm each sense is determined for each word individually without considering the senses assigned to neighboring words. Based on the results of the previous experiments carried out on graph based and similarity based approaches it's been observed that graph based approaches outperform similarity based ones, by a significant margin. [5]

A clear advantage of graph based WSD systems is that the entire UMLS Knowledge Base can be used during the disambiguation by propagating information through the graph [1].

2.4 Applications of WSD

2.4.1 Machine Translation/Word Understanding:

To identify exact translation of a word in a particular context is an extremely difficult task to perform automatically. WSD has been considered as a major task which needs to be solved to enable an accurate machine translation, this is because it is widely known that disambiguation of words in a sentence can help choose better candidates as depending on the context words can have totally different translations. Even though WSD disambiguation is very difficult to implement and some other methods have been proposed it still is the best option [11].

2.4.2 Data Retrieval:

Express semantics are not used to tight down records, which are not pertinent to the client by even the most progressive Internet searchers. The execution issues and the extensive overhead that may come about because of the enormous knowledge base scan is the real reason that WSD has not contributed fundamentally to data recovery truly. However, with better routines to execute WSD it could be utilized to precisely offer what the client asked for, an exact disambiguation of the report database alongside the disambiguation of the questioned words will encourage the determination of just those archives, which are really required [11].

2.4.3 Content Analytics:

Examination of content as for thoughts, topics, tones, and so on can profit by WSD utilized as a part of substance investigation space. Consider the sample of Blogger, it contains such a variety of online journals, and their number is expanding quickly. Content investigation utilizing WSD can help as a part of characterization of information with according to client necessities [11].

2.4.4 Semantic Network/Web:

Semantic Web is only a cooperative development by World Wide Web Consortium to urge website pages to incorporate semantic substance into their site pages to change over the right now existing unstructured or semi organized archives into a web of information. All the aforementioned strategies can be used to accomplish this vision and consequently WSD assumes a critical part in accomplishing it [11].

3 PROJECT DESIGN

3.1 Definition

3.1.1 Problem Formulation

Given an input to the system as a set of clinical notes or a clinical discharge summary note, output the most correct disambiguated sense for the detected medical words or abbreviations using graph based unsupervised word sense disambiguation techniques implemented on Apache Spark.

3.1.2 Terminology

The following terms are widely used in the report:

- *Entity*: In this project, entity represents an object with unique id and properties.
- *Clinical Note*: an ICU patient report given by a physician.
- *WSD*: Word Sense Disambiguation.
- *Similarity*: Denotes the relevancy between an entity also known as a medical term or an abbreviation in the knowledgebase and abbreviations detected in the input clinical note.
- *Context Words*: Bag words surrounded around the medical abbreviation or the medical word, which has to be disambiguated.
- Resilient Distributed Datasets(RDD): The basic abstraction in Apache Spark

3.2 Technology

3.2.1 Apache Spark

Apache Spark is an open source cluster-computing framework developed at the AMPLab of UCBerkley. By distinctively performing in-memory data processing Spark sets itself out of Hadoop open-source community. Spark is not build on the fundamental blocks of Map and Reduce. Spark provides real-time analytics by processing large volume of stream data. There are several advantages of Spark as compared to other big data and MapReduce technologies namely Hadoop and Storm.

Spark distributes all the actions as applications across the cluster and runs them as independent sets of processes. The SparkContext residing in the 'driver program' of a spark job controls the cluster. Driver program also hold the main function that triggers the job and starts execution of non-slave operations. In a compatible cluster environment like Yarn also known as Yet Another Resource Negotiator, Spark registers executors on nodes in cluster and it sends the application code. After all the executors are registered, SparkContext send divides the batch in to set of tasks and distributes then across the nodes. Figure 3 shows the basic building blocks of how prime components of Spark works together. The processes are active as long as the corresponding application is running. The architecture is logically divided into two stages as scheduling stage and the executor stage.

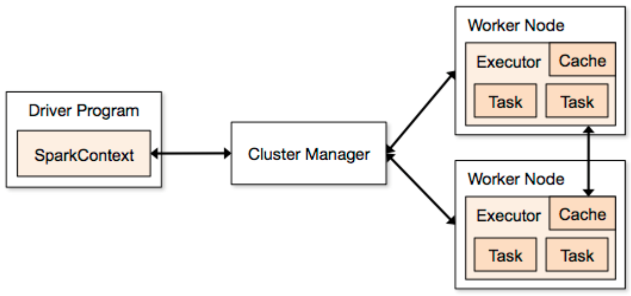


Figure 3: Apache Spark Architecture

The driver schedules every individual task in the scheduling stage and then executor stage runs the applications in different JVMs. Performance of Spark heavily depends on the cluster manager component. Executor processes communicate and share data chunk reference through cluster manager. It helps the worker nodes in the cluster to acquire resources on the cluster and it essentially shares the resources of cluster amongst Spark Applications and it also assist the driver by creating the executors. For solutions like the one implemented in this project, the in-memory processing of Apache Spark assist significantly in improving the performance along with the lazy evaluation of the large data queries across the cluster. Spark provides support for Structured Query Language through Spark SQL to streamline data querying process on the data stored in RDDs and the external sources like Hive, HBase. Through Spark SQL the underlying RDD data in conveniently abstracted as relational tables.

Spark also provides support for Graph processing through GraphX, API for graphs and graph-parallel computation. It essentially extends the RDD abstraction by introducing the RDD Property Graph, a directed multi-graph with vertex, edges and attached properties.

3.2.2 Amazon Web Services EMR

Elastic Map Reduce (EMR) is based on Hadoop, that supports processing of data in distributed environment. With the MapReduce framework it allows the developers process massive amount of structured and unstructured data in parallel. EMR also support Spark clusters along with Hadoop. It processes data across the cluster of virtual servers. Also, it provides the capability to scale up and scale down the cluster resources depending on processing requirement.

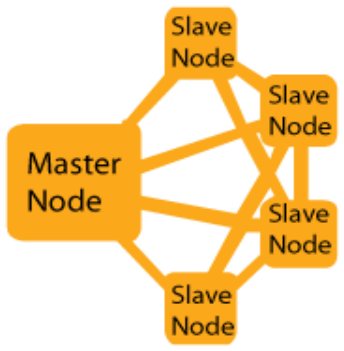


Figure 4: AWS EMR Logical Model

3.2.3 Amazon Web Service S3

Simple Storage Service is web-based scalable, high-speed service for online data backup and archiving. S3 support upload, store and download of almost any file type up to five gigabytes in size. Redundant servers are utilized for storing the data across multiple data centers. It targets to maximize the scaling capabilities and enable applications to grow with storage requirements.

3.3 Methods

3.3.1 Jaccard Similarity

Jaccard Similarity is a statistic measure used to compare the similarity and dissimilarity of sets. Similarity is defined as set intersection size divided union size. This similarity measure is effective to represent similarity between two documents or collection of terms.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Jaccard similarity measure is useful for similarity computation between set of words in word sense disambiguation problem.

3.3.2 Betweenness Centrality

Betweenness centrality measure is a metric to calculate the highest score amongst the nodes in the graph of which the correct sense for that particular word has to be determined. The betweenness of node v is calculated as the fraction of shortest paths between node pairs that pass through v. Formally betweenness is defined as:

$$betweenness(v) = \sum_{s,t \in V : s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Where σ_{st} is the number of shortest paths from s to t, and σ_{st} (v) the number of shortest paths from s to t that pass through vertex v. The intuition behind betweenness is that a node is important if it is involved in a large number of paths compared to the total set of paths.

4 IMPLEMENTATION

This is a summary on the implementation of the Word Sense Disambiguation System on Apache Spark. This section covers essential implementation details with original designs and source code snippets for reference.

4.1 Apache Spark Setup

4.1.1 Installation

This section lists the tools and technology setup that was required along with Spark for development and deployment activities throughout the project.

- Apache Spark 1.5.0 with Hadoop 2.6
- Scala 2.10.5
- Java 1.8
- AWS CLI

Selection of required version for all the installations was performed by analyzing the version dependency across listed items.

4.1.2 Configuration

Development and deployment were done in two different configurations as discussed below. AWS Simple Storage Service has been utilized to store all the data required in the application throughout all the processes.

- Standalone

  Development activities were carried out with Spark in standalone mode on a system with following primary specifications.

  - Cluster on a machine with Intel i7 Processor and 16 Gigabytes RAM

    - Master: local [4]

    - Driver Memory: 4g

    - Executor Memory: 2g

    - Spark Configured for AWS S3 Access: Yes

- Cluster

  Deployment of the developed application was done in cluster setup on Elastic Map Reduce service provided by Amazon Web Services (AWS).

- EMR 4.1.0 with Apache Spark 1.5.0 with Hadoop 2.6
- Master: 1, Slaves: 2
- EC2 instance (m3.xlarge) with 4 CPU Cores and 16 GB RAM each.
- AWS CloudWatch Enabled: Yes
- Spark Configured for AWS S3 Access: Yes

4.2 Algorithm

This section describes algorithm used to implement word sense disambiguation of abbreviations for clinical notes. The algorithm uses UMLS Metathesaurus as a graph K of CUI's also known as Concept Unique Identifiers for each medical term in the UMLS. There are several tables that are part of the Metathesaurus, our implementation makes use of two primary tables: MRREL and MRCONSO. The MRCONSO table is the primary table, which contains one row per file and has detailed meaning of each unique string. It implies that every combination of CUI and STR has only one row in the table. The MRREL table contains all the relations between a Metathesarus string and the CUI associated with it, so essentially it contains the relation mapping details between concepts. The relations are defined of ten different types, which range from narrower relations to broader relations. For performance consideration and we have focused on following six relation types:

- PAR, the parent relation
- CHD, the child relation
- RB, the broader relation
- RN, the narrower relations
- SIB, the sibling relation
- RO, the other relation

Following section elaborates on the essential details of the algorithm used for implementation along with significance of each step and its outcome in general WSD problem solving.

```
WordSenseDisambiguate ($K$, $W$, $t$, $s$, $A$)
1:  let $V$ = {UMLS concept of $W_l$ | $l$ = ($t$-1..$t$-$s$) ∪ ($t$+1..$t$+$s$))}
2:  let $V$ = $V$ ∪ $A$
3:  for each $v$ in $V$ do
4:      $X$ = DFS($K$,$v$,$p$)
5:      for each $x$ in $X$ do
6:          if ($x$ not in $V$)
7:              let $V$ = $V$ ∪{$x$}
8           end if
9:      end for
10: end for
11: let $E$ = GetEdges($V$,$K$)
12: let VRanks = Betweenness($V$,$E$)
13: let $m$ = maximum{ VRanks ($a$) | $a$ in $V$ and $a$ in $A$}
14: return $m$

DFS($K$,$v$,$p$)
1:  return the set of nodes encountered when performing depth-first search
starting from node $v$ in the graph $K$ at a maximum depth $p$.

GetEdges($V$,$K$)
1: return the set of edges in graph $K$ that interconnect all nodes in the $V$ set.

Betweenness ($V$,$E$)
1: return a set of all nodes in $V$ with their betweeness metric
```
[6]

This algorithm considers following three major steps in the execution:

- Tokenization with WSD Candidate Selection and Bag of Words Collection.
- Neighbor collection for all the identified token concepts on Graph.
- Similarity/Betweenness calculation on collection of neighboring nodes.

Following is the description of each step in the algorithm for better understanding of how it works in general case with the data in consideration.

1. An unstructured clinical note is parsed to valid set of collect tokens.
2. Parsed tokens are verified for abbreviation detection and context word selection.

3. Abbreviation detection is performed by filtering the tokens against abbreviation listing created with unique set of abbreviations from three overlapping medical abbreviation dictionaries: ADAM, BERMAN and LRABR.
4. For each detected abbreviation, bag of words i.e. context words are collected based on specific size of window.
5. Each set of abbreviations and corresponding context words in the document are then processed together by scanning each word against the UMLS graph of concepts for identifying the CUIs for the abbreviations and context words.
6. In the graph scan, edges will be collected where either the abbreviation or the context words are on the source or destination side of the edge.
7. Traversing through the graph in Depth First Search mode purely performs the selection of edges to collect each edge triplet. This will result in required collection of CUIs that are specific to a set of abbreviation and context words.
8. The set of all the CUIs collected in previous step are then processed through Betweenness Calculation by building sub-graph of CUI nodes and ranking which of the abbreviation CUIs output the highest betweenness score.
9. All the CUIs for each unique abbreviations with maximum betweenness score are then considered as the most correct meaning of the abbreviation based on the context words it is surrounded with.

4.2.1 Algorithm Implementation Steps in WSD on Apache Spark

Preparation Phase:

- Construct GraphX graph from UMLS - MRREL. CUIs will be the Vertex and Relations between CUIs will be edges on the graph.
- Build and Load abbreviation dictionary along with listing of tokens like measurements, stop words etc. that will assist parsing process.

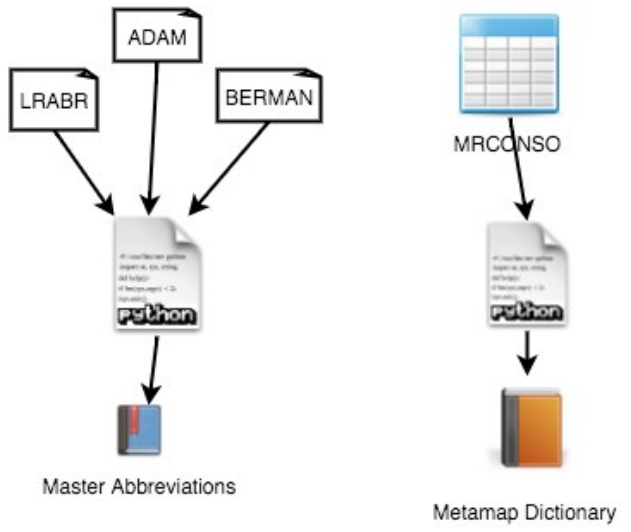


Figure 5: Flow of how data is fetched from medical dictionaries

Execution Phase:

1. For each clinical note input we parse the documents to get following

- W, a sequence of n words, representing the text containing the word to be disambiguated with t, an index in W pointing to the word to disambiguate.
- s, a window size of the words before and after t.
- A, a set of plausible senses for the word being disambiguated.

2. For each abbreviation, we use the simulated in-memory Metamap dictionary to query the CUIs and build a list of Abbreviation to CUI mapping.

3. Abbreviations with single resulting CUI are considered as already disambiguated as they refer to only one meaning and the meaning is consistent across all senses.

4. Abbreviations for which there are no CUIs identified are classified as abbreviations that cannot be disambiguated with the available metadata and dictionaries.

5. Abbreviations with more than one identified CUIs are considered as the disambiguation candidates. For all the disambiguation candidate abbreviation,

the simulated metamap will be queries to get the CUIs for all the context words as the CUIs for abbreviations were already collected in step #2 above.

6. For each identified unique CUI of Abbreviation and Context Words we travers the Graph built in preparation phase using GraphX and UMLS MRREL. Traversing the graph in Depth First Search fashion collects neighboring nodes for each CUI. A map of CUI and neighboring nodes is then constructed and broadcasted across the clusters for quick access query on each node.

7. After all the neighboring nodes are collected for each CUI from a set of abbreviation and it's corresponding context words we perform the next step to identify the most relevant CUI in two ways: Jaccard Similarity computation and Betweenness Computation.

8. Jaccard similarity computation is an additional adopted approach beyond the scope of the original algorithm in this report. It is observed that similarity computation yields almost identical results to the betweenness computation. Hence, this implementation has been made configurable to provide choice selecting either of the computation methods.

9. Based on above discussed two computation methods we derive one CUI for each abbreviation which either fell on the way of all the context word neighbor nodes for it in Betweenness Computation or the sets of abbreviation CUI neighbors and Context Words CUI neighbors matched the most with Jaccard Similarity.

10. Final result will be set of abbreviations per document with their CUI and concept attached with CUI. This will be consolidated output of preprocess step where we eliminated the non-ambiguous candidates and the output of second step where we actually disambiguated the disambiguate candidate.

Following diagram depicts implementation of each step discussed previously.

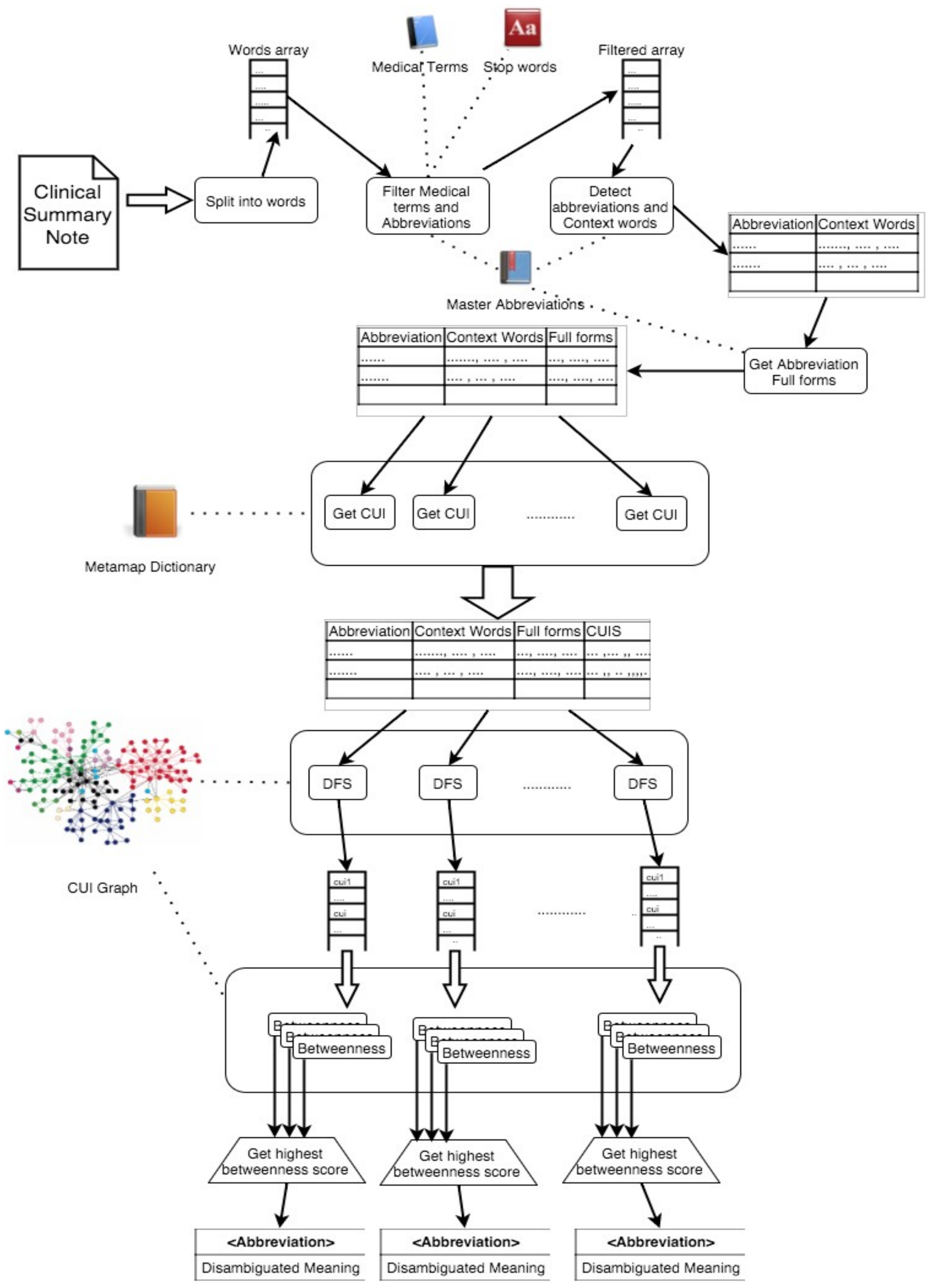


Figure 6: Data Flow Functional Architecture Diagram

4.3 Challenges

1. Simulation of Metamap

- Metamap is a UMLS service that provides CUIs for queries that contains strings to looks for in the UMLS MRCONSO.
- Metamap exposes REST APIS through which other application programs can query and get the CUIs for the words.
- This project has a critical dependency on the Metamap to access metamap through REST APIs up to 800 times for each clinical note.
- Hence, it was challenge to find alternative solution eliminate performance bottleneck and dependency on an external service by maintaining same level of correctness in query results.
- Storing the MRCONSO in an external database and querying it extensively would have improved performance and removed the dependency from Metamap. However, it was not an ideal solution.
- The solution was identified as SparkSQL, which helped us in loading the aggregated MRCONSO table and distributing it across the nodes.
- Hence, all the queries were performed on in-memory data with reducing Spark Read shuffles by querying once and broadcasting the mapped results across the nodes, this essentially improved the overall performance of each query.

2. Read Shuffle due to GraphX Scanning and Spark SQL Queries
    - This was identified as the biggest challenge to performance optimization.
    - It was practical to load the MRCONSO in SPARK SQL and MRREL in GraphX. However, querying then hundreds of time for each clinical note was impractical considering the volume of data that is distributed across nodes.
    - A single query would result in huge number of read shuffles causing data to move across nodes and heavily affect the performance of the application. However, Querying the Spark SQL and GraphX from driver was not as expensive as from worker nodes.
    - Hence, the identified solution was to build a list of unique CUIs from each clinical note and send it back to Master
    - Driver program would collect the lists of neighbors in case of scanning neighbors and list of meaning strings in case of querying the meaning for CUIs.
    - This collected list will be broadcast across nodes to make it available for distributed processing of each set of abbreviation CUIs and Context Word CUIs identified in previous step.

3. Identifying the best relation types to consider for building graph
   - There were ten relation types between CUIs in MRREL from which the graph was required to build. And the performance of graph was dependent on it.
   - Hence, identified solution was to build the graph with selected six primary narrower relationships. This helped in reducing the size of graph and improved the overall performance of traversing the graph.

4.4 Optimization using Apache Spark

Before describing the optimization framework used in project, following are the feature of Spark on high-level that helped to enhance performance of the project.

4.4.1 Fast in-memory processing

As the volume of data is growing exponentially, there is a need for excellent processing performance in all types of applications. With the primary abstraction of Resilient Distributed Datasets, Spark efficiently distributes the data across nodes in the cluster and performs computation effectively by distributing tasks for better parallel processing by benefitting with the data locality across the nodes. In this project, all the required metadata is loaded in-memory at specific required stages. This enabled faster parsing of clinical notes and abbreviation detection. Also, as the data is consistently maintained across the nodes, passing the data from one stage of RDD to the next stage did not cause much overhead.

4.4.2 Query Processing with SparkSQL

Spark’s support for relational data with its Spark SQL dataframe abstraction over the RDD enables fast querying and analytics of data stored in-memory. Having SQL query like capability on in-memory data improved the overall performance across the application where there is frequent need of selecting specific data from millions of rows.

In this project, we simulated the functioning of metamap ensuring to gain similar results. Spark SQL was highly effective in replicating the MRCONSO relational data in memory and providing real-time results to hundreds of queries for CUI retrieval.

4.4.3 Spark GraphX

Sparks’ GraphX API helps to store the data in-memory in the form of vertex and edges by distributed it across the cluster. It enables much faster traversing and computation on the graph data as compared to using any external Graph APIs.

Loading the UMLS MRREL data in-memory as a graph was critical for optimized implementation of the chosen algorithm. The GraphX API allowed us to load the complete MRREL data as graph nodes and edges. It also enables us to traverse the graph and load node-edge-node triplets with high performance. And as the graph can be persisted and un-persisted as required, it was very useful to persist data before the stage where graph traversing was needed to perform and unresisting it after operations are over and graphs is not required for any further processing. GraphX API also let us query just the nodes and edges individually; this was very helpful in specifically getting the neighboring nodes and filtering the edges on type of relationships.

4.4.4 Broadcast Objects

Through Broadcast feature, Spark allows to store the copy of frequently used data across the nodes. This is very effective in improving the performance of the application as the data is local to the node and no network traffic or read shuffle would take place while a worker node tries to read the data that is broadcasted through the SparkContext previously. Also it is possible to un-persist distributed objects required, this allows freeing up the memory when the data is not required anymore. Metadata objects required for parsing, abbreviation detection and the objects holing graph neighbors were broadcast to improve the performance of each node.

4.5 Execution Framework

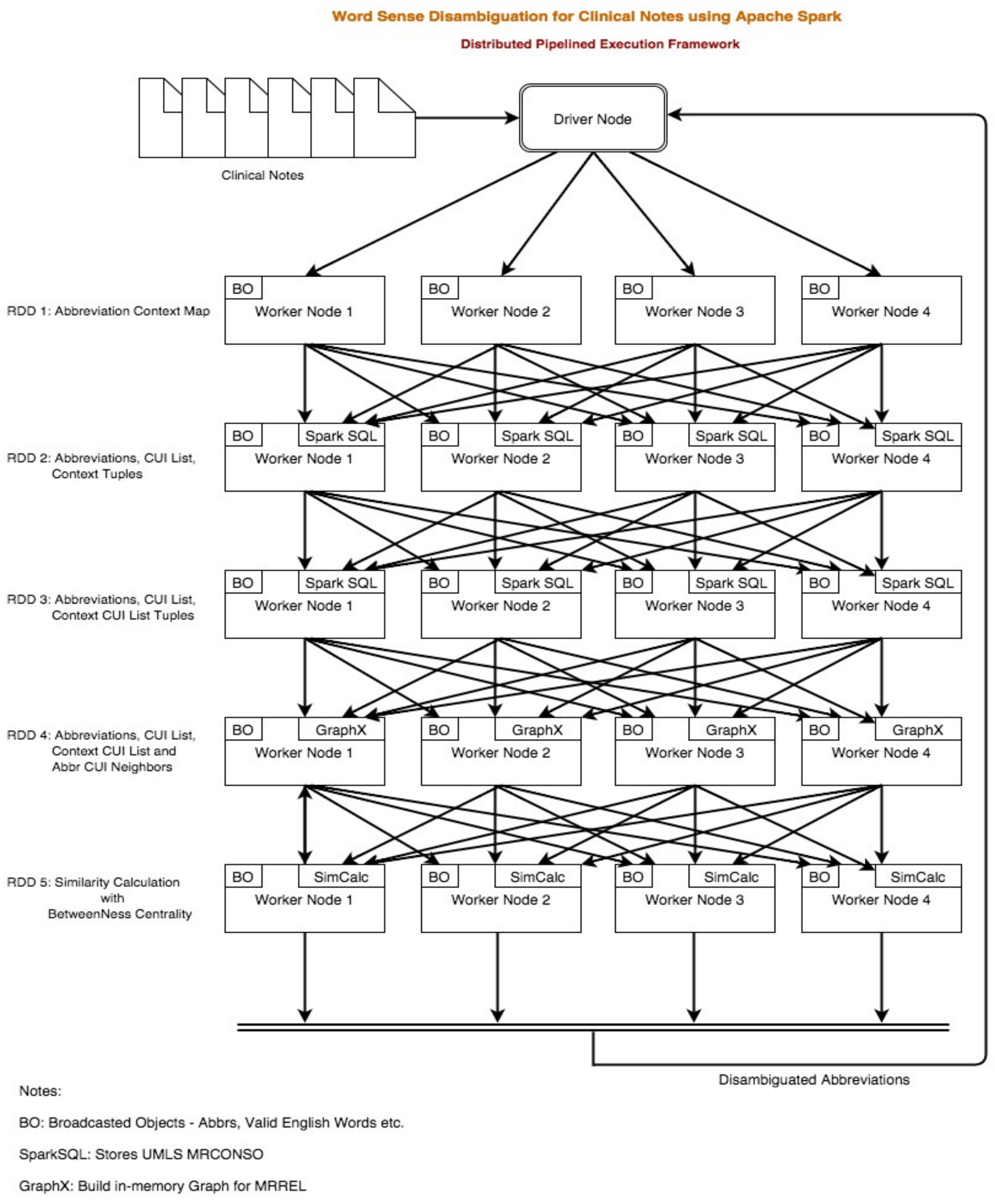
This section describes execution framework that enabled a highly optimized implementation of algorithm with features of Spark as discussed above.

4.5.1 Distributed Pipeline Execution Framework

The pipelined execution framework assumes that a multi-stage execution would progress by processing on the output of previous stage. Every stage of processing will be a set of tasks distributed across executors running on the worker nodes in the cluster. Distributed Pipeline refers to the use of data locality for multi-stage processing of the data on that resides on the same worker node throughout all the processing stages.

In this project, the processing is starts by distributing the Incoming input notes to workers for tokenization and further processing. The complete execution is divided into two major stages as Preprocess Stage and the Disambiguation Stage. Both the stages are collection of multiple processes that internally pass the output of one process to the next for the completed output of the overall stage. It is ensured that data present on the same node is used in all the processes of a stage. Hence, this framework ensures minimal read shuffles that generally affect the performance of worker nodes and the cluster. Since the output of each stage remains in the RDDs, the output of one stage is mapped with a particular function generates a new RDD through which Spark ensure that the data present on a worker remains on the same worker. This allows the framework process with required performance, as the data locality is benefits by ensuring that data is not moved across nodes if not required. It eventually results in pipelined approach across processes running on the individual worker nodes.

Finally, as per the term, distributed pipeline, each worker node data would be available to subsequent stages unless the implemented algorithm demands shuffling. Figure 7 depicts the Distributed Pipelined framework as discussed above.



Word Sense Disambiguation for Clinical Notes using Apache Spark
Distributed Pipelined Execution Framework
Clinical Notes
Driver Node
RDD 1: Abbreviation Context Map
RDD 2: Abbreviations, CUI List, Context Tuples
RDD 3: Abbreviations, CUI List, Context CUI List Tuples
RDD 4: Abbreviations, CUI List, Context CUI List and Abbr CUI Neighbors
RDD 5: Similarity Calculation with BetweenNess Centrality
BO
Spark SQL
GraphX
SimCalc
Worker Node 1
Worker Node 2
Worker Node 3
Worker Node 4
Disambiguated Abbreviations
Notes:
BO: Broadcasted Objects - Abbrs, Valid English Words etc.
SparkSQL: Stores UMLS MRCONSO
GraphX: Build in-memory Graph for MRREL

Figure 7: Distributed Pipelined Execution Framework

5 PERFORMANCE

This section discusses the accuracy and performance of the implemented application along with optimization techniques that were applied to optimize the solution even further for faster and all in-memory processing.

5.1 Application Performance

This section discusses the performance of the implementation observed for individual processing of clinical notes from a set. Total performance of the application is dependent on several factors when running on Spark. The major factors the contribute to total run time of an application for a single clinical note processing are:

5.1.1 Core Processing Time

This is the time taken perform following tasks

- Parse clinical note, create list of tokens and detect abbreviations
- Select context words and build Abbreviation-Context map
- Fetch CUIs and Build List of CUIs for each Abbreviations
- Detect Abbreviations that are readily disambiguated
- Detect Abbreviations that cannot be disambiguated
- Detect Abbreviations that are real candidates for disambiguation
- Detect CUIs and build list for context words of candidate abbreviations
- Disambiguation of detected abbreviation candidate

Output of core processing is as follows:

- List of Abbreviations with only one associated CUI.
- List of Abbreviations with no associated CUI.
- List of Abbreviation with more than one associated CUI.
- Finally, the actual list of disambiguated abbreviations with CUI and meaning

5.1.2 Graph Scanning Time

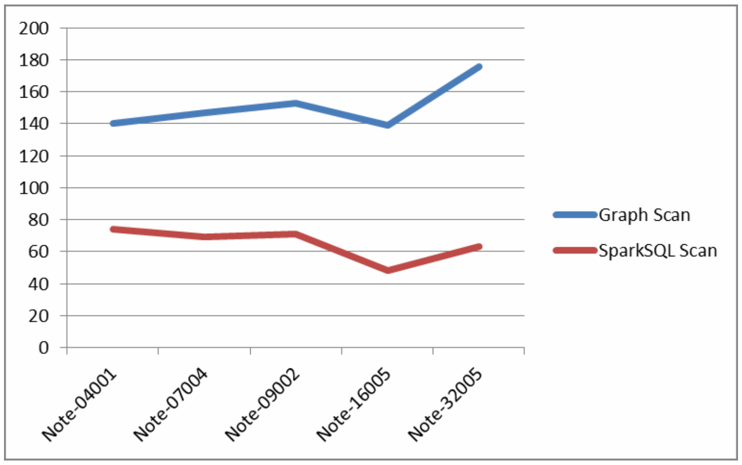
This is total consolidated time taken to traverse the graph for finding required CUI nodes and collecting their neighbors. This time heavily depends on total number of nodes to be accessed, total neighbors attached to each node and the distribution of graph data across the cluster. In case of this application, the nodes and neighbors are collected by master, which reduces the read shuffles across the nodes significantly. Also, the performance of graph traversal was improved substantially by loading the node and neighbor maps in memory through broadcasted objects. In this application the graph traversing is done only once per clinical note. And if multiple clinical notes are fed at once, the graph traversal will be performed once for all the detected abbreviation CUIs across all the documents.

5.1.3 SparkSQL Query Time

This is total consolidated time taken to query the SparkSQL table that contains the Abbreviation to CUI mappings that are essential at every stage of disambiguation process. Similar to graph traversing process, the querying is also performed in batch. A smart approach was taken to aggregate all the CUI entries for specific abbreviations and build a compact list of CUIs that can be broadcasted to the worker nodes. Hence the total time to query is cumulative of time taken to read the aggregated list, broadcast it and query it during the processing. The queried results are made available to worker nodes in term of a Abbreviation to CUI List map which is used in the first step of disambiguation to detect abbreviations with single CUI, no CUI or multiple CUIs. And similar map will be used to fetch the CUI meaning in the final stage where the disambiguated CUIs are being mapped with their associated meaning to present the output with corresponding abbreviation. Following table shows the summary of processing time in seconds as per above discussion for five different clinical notes.

Sr. No.	Clinical Note	Total Application Run Time	Core Processing Time	Graph Scanning Time	SparkSQL Query Time
1	NOTEEVENTS-04001.txt	**218s**	**101s**	**89s**	**28s**
2	NOTEEVENTS-07004.txt	**240s**	**134s**	**84s**	**22s**
3	NOTEEVENTS-09002.txt	**205s**	**98s**	**76s**	**31s**
4	NOTEEVENTS-16005.txt	**212s**	**102s**	**83s**	**27s**
5	NOTEEVENTS-32005.txt	**183s**	**91s**	**71s**	**21s**

Table 1: Summary of Processing Time for set of Clinical Notes



Y-Axis – Time in Seconds X-Axis – Clinical Notes

Figure 8: Average Processing Time for Graph Traversal and Query Retrieval

Below table shows the runtime performance of the application for single clinical note processing with Spark based application processing parameters.

Runtime Performance	~ Seconds
Job Initiation Time	2
Loading Dictionaries In-Memory and Broadcast across Workers in Cluster	6
Loading UMLS graph of 28 million records	27
Traversing GraphX and Building GraphMap	84
Time for Map Task - RDD Map Stages - Preprocessing & Disambiguation	79
Time for Reduce Task - RDD Reduce Stages - RDD Collect Stages	49
Time for loading MRCONSO in Spark SQL	34
Time for SparkSQL Query for Disambiguated CUI Meaning	22
Time for writing back to disk	59

Table 2: Runtime Performance for Single Clinical Note

5.2 Disambiguation Results Summary

This section discusses the summary of disambiguation results for abbreviations detected in a set of clinical notes. The results are segregated as per clinical notes for clarity. Following the main components considered presenting the results.

- Total Abbreviations Detected

This factor determines the total number of abbreviation detected in a clinical note. As per multiple runs on multiple clinical notes, it is observed that detected abbreviations are actually the total present abbreviations in the clinical note. Hence the parsing of tokens and abbreviation detection has excellent accuracy

- Total Abbreviations Disambiguated

  It is the total number of abbreviations that were disambiguated from the total abbreviation detected abbreviations. This is a consolidated count of abbreviations that are disambiguated correctly and the abbreviations that are disambiguated incorrectly.

- Accuracy - Total Abbreviations Disambiguated Correctly

  This is the total count of abbreviations that were disambiguated correctly which directly refer the accuracy of overall disambiguation process. The correctness of results is verified against the relevant of CUI with the context by analyzing clinical note and UMLS data.

The additional components of Disambiguation Results are the as below:

- Abbreviations with Single associated CUI

  The abbreviations that were mapped to single CUI from the Metamap, were considered as the straight forward disambiguated abbreviations as the mapped single CUI associate the abbreviation a single meaning that eliminates the ambiguity for abbreviation meaning. These abbreviations contribute to the total number of correct disambiguation as the results fall into true positive category.

- Abbreviations with No associated CUI:

  The abbreviations that did not map with any of the CUIs in the Metamap cannot be considered as the candidates for disambiguation. Hence, in the preprocessing stage, all such abbreviations are filtered to avoid unnecessary processing overhead in further stages. These abbreviations also contribute to the complete list of abbreviations that were not disambiguated, which helps to understand the impact of reference data on the results.

Following table shows the summary disambiguation results per above discussion for five different clinical notes.

No.	**Clinical Note**	**Total Abbreviations Detected**	**Total Abbreviations Disambiguated**	**Accuracy [Correct Disambiguation]**
1	NOTEEVENTS-04001.txt	**98**	**89**	**71**
2	NOTEEVENTS-07004.txt	**141**	**92**	**79**
3	NOTEEVENTS-09002.txt	**90**	**53**	**49**
4	NOTEEVENTS-16005.txt	**74**	**62**	**56**
5	NOTEEVENTS-32005.txt	**102**	**53**	**45**

Table 2: Summary of Disambiguation Results for set of Clinical Notes

5.3 Multiple Stage Results for Detected Abbreviations

5.3.1 Preprocessing Stage

The preprocessing stage generates three lists of outputs. Abbreviations with single CUI, Abbreviations with No CUIs mapped and Abbreviations with multiple CUIs mapped. The later ones are the real candidates for disambiguation process. Further in this section discusses the multiple intermediate outputs that are generated as part of preprocessing stage.

- Abbreviations with Single CUI

The abbreviations with single CUI are considered as elements with single meaning and are not required to go through the complete disambiguation process. The abbreviations are already disambiguated considering there is now ambiguity due to the clear singular mapping with reference data. Following table shows the list of abbreviations that were collected from the test runs

Abbreviation	Retrieved CUI from Metamap	Meaning
CHF	C0018802	CONGESTIVE HEART FAILURE
AMT	C1412390	Amount
MR.	C2347167	Mr. - Title
PPM	C0439187	Part per Million
SVC	C0231957	Slow Vital Capacity
NEG	C3853545	Negative
N/C	C2349138	Volt per Meter
CFU	C0553561	Colony Forming Unit
FOCI	C0205234	Focal
DR.	C2348314	Doctor - Title

Table 3: Abbreviations with Single CUI

- Abbreviations with No CUI Mapped

Following is the list of abbreviations to which no CUI from Metamap was mapped in the preprocessing stage. All these abbreviations were not considered for further disambiguation processing.

Abbreviation
SPO2
OSH
HCT
TOL
F
OCC
U/O
S/P
Y/O
AOX

Table 4: Abbreviations with No CUI Mapped

- Abbreviations with Multiple CUIs Mapped

Following is the list of candidate abbreviations that were passed to the execution stage for disambiguation. The output of preprocessing stage for these abbreviations is three fold. List of abbreviations with their context words, list of CUIs for each abbreviation and list of CUIs for each context word. These all elements are aggregated as a tuple for each abbreviation, which allows distributing the tuples across nodes without the need of maintaining all the abbreviations from a clinical note together on a single node.

Abbreviation	Context Words
ALT	mildly, elevated, 46, alkaline, phosphatase, 52, ast, checked
AST	mildly, elevated, 52, alt, checked, transaminases, liver, tegretol, started
UTI	positive, blood, cultures, change, 1158**], unit, [**hospital, osh
GU	normal, male, testes, descended, organomegaly, soft, abd
HBSAG	neg, rpr, nr, ri, gbs, ab, pos, , pns:
BP	81/49, 61, , temp, 100.0, 60, rr, 136, hr,
RPR	nr, ri, gbs, negative., , neg, hbsag, ab, pos
URI	fever, days, prior, delivery, resolved, throat, sore, , lesions., herpes
CSF	usual, studies, pcr, hsv., will, clear., appeared, fluid, -, nnp
GI	bleed, major, surgical, invasive, procedure:, complaint:, chief, 203**

Table 5: Abbreviations with Context Words

Abbreviation	Retrieved CUIs from Metamap
ALT	C0001899, C0201836, C0376147, C2257651
AST	C0004002, C0201899, C0242192,C1420113
UTI	C0042029, C1412376, C0077906
GU	C0018309, C0042066, C2709258
HBSAG	C0019168, C0201477, C0796320
BP	C2986841, C0057191, C0037623, C0005823, C0005824
RPR	C0201405, C1705631
URI	C0041912, C1421895, C1548524, C1548524, C3272713
CSF	C0007806, C0009392, C0079460, C3540512
GI	C1136206, C1415142, C0017187, C0017540, C1553044, C1551090, C0521362

Table 6: Abbreviations with mapped CUIs from Metamap

Similar to above there is additional list of context word CUIs. The lists are Abbreviation CUIs and Context CUIs for each abbreviation are essential for the next stage where graph traversing and further processing will be done based on each CUI from these lists.

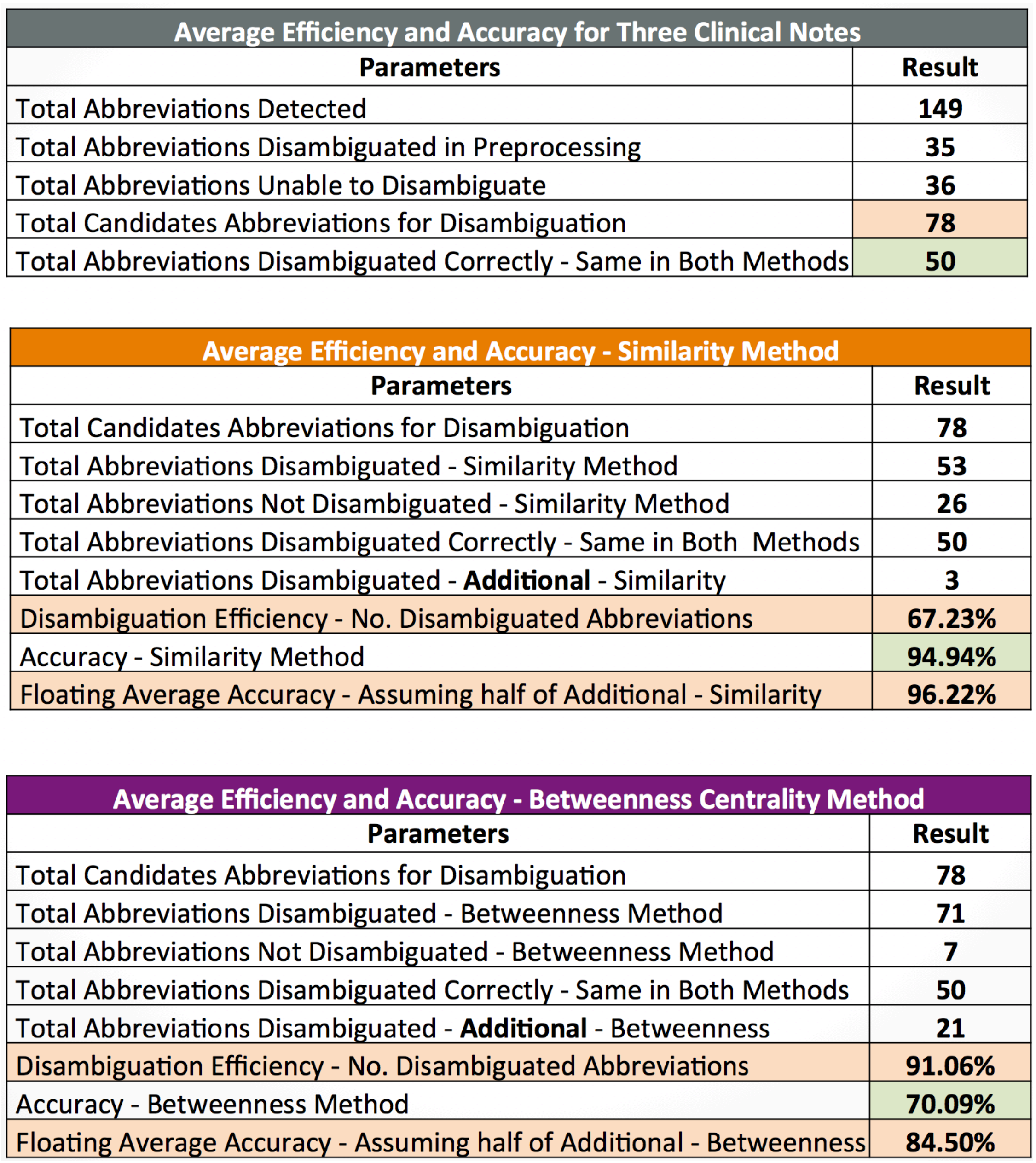
5.3.2 Execution Stage – Disambiguation

In this stage, the output processed by the preprocessing stage is picked up for traversing through the graph and getting the neighboring nodes for each CUI of the abbreviation and for each CUI of the context words. After getting all the neighboring CUIs, the most relevant CUI for the abbreviation is determined by applying the Betweenness Centrality and Jaccard Similarity Methods as discussed in previous sections of this report. In essence, this implementation performs two steps for disambiguation, 'Word Sense Disambiguation' in which maximum abbreviations are disambiguated based on context words surrounding them. The abbreviations that were not disambiguated because of non-relevant context words supplied to higher- l e v e l implementation of Document Sense Disambiguation which tries to determine the 'sense' of a word from the whole document as the context.

Abbreviation	Disambiguated CUI	Meaning
ALT	**C0376147**	serum glutamate pyruvate transaminase
AST	**C0004002**	Aspartate Transaminase
UTI	**C0042029**	Urinary Tract Infection
GU	**C0042066**	GENITOURINARY
HBSAG	**C0201477**	Hepatitis B Virus Surface Antigen
BP	**C0005823**	Blood Pressure
RPR	**C0201405**	Rapid Plasma Reagin
URI	**C0041912**	Upper Respiratory Tract Infection
CSF	**C0007806**	Cerebrospinal Fluid
GI	**C0521362**	gastrointestinal

Table 7: Examples of Disambiguated Results with Highest Accuracy

Following are the average accuracy comparison results computed for disambiguation performed on multiple notes with similarity and betweenness centrality measures. The ultimate average accuracy of the application i.e. 82.5% is calculated by considering the average accuracy of each method.



Average Efficiency and Accuracy for Three Clinical Notes	
Parameters	**Result**
Total Abbreviations Detected	**149**
Total Abbreviations Disambiguated in Preprocessing	**35**
Total Abbreviations Unable to Disambiguate	**36**
Total Candidates Abbreviations for Disambiguation	**78**
Total Abbreviations Disambiguated Correctly - Same in Both Methods	**50**

Average Efficiency and Accuracy - Similarity Method	
Parameters	**Result**
Total Candidates Abbreviations for Disambiguation	**78**
Total Abbreviations Disambiguated - Similarity Method	**53**
Total Abbreviations Not Disambiguated - Similarity Method	**26**
Total Abbreviations Disambiguated Correctly - Same in Both Methods	**50**
Total Abbreviations Disambiguated - **Additional** - Similarity	**3**
Disambiguation Efficiency - No. Disambiguated Abbreviations	**67.23%**
Accuracy - Similarity Method	**94.94%**
Floating Average Accuracy - Assuming half of Additional - Similarity	**96.22%**

Average Efficiency and Accuracy - Betweenness Centrality Method	
Parameters	**Result**
Total Candidates Abbreviations for Disambiguation	**78**
Total Abbreviations Disambiguated - Betweenness Method	**71**
Total Abbreviations Not Disambiguated - Betweenness Method	**7**
Total Abbreviations Disambiguated Correctly - Same in Both Methods	**50**
Total Abbreviations Disambiguated - **Additional** - Betweenness	**21**
Disambiguation Efficiency - No. Disambiguated Abbreviations	**91.06%**
Accuracy - Betweenness Method	**70.09%**
Floating Average Accuracy - Assuming half of Additional - Betweenness	**84.50%**

Figure 9: Disambiguation Average Accuracy Results

5.4 Examples

5.4.1 Example 1:

Clinical Note: NOTEEVENTS-07004

Abbreviation: URI

Sentence from input Clinical Note:

She had a sore throat without URI or fever for several days prior to delivery that resolved right after delivery.

Detected Context Words:

fever, days, prior, delivery, resolved, throat, sore, , lesions., herpes

Total CUIs for this abbreviation in Metamap with their Meaning:

C0041912 - Upper Respiratory Tract Infections

C1421895 - UNCONVENTIONAL PREFOLDIN RPB5 INTERACTOR

C1548524 - Uniform Resource Identifier

C1548524 - Uniform Resource Identifier

C3272713 - Chromosome 19 Open Reading Frame 2 wt Allele

Disambiguated Accurate CUI: C0041912

Conclusion:

To validate the correction of above result, we verified the abbreviation and context word relation in clinical documents on valid medical websites.

Primary website used for result validation:

http://www.hopkinsmedicine.org/healthlibrary/conditions/pediatrics/upper_respiratory_infection_uri_or_common_cold_90,P02966/

Description related to URI in Clinical Text from Website:

An upper respiratory infection (URI), also known as the common cold, is one of the most common illnesses, leading to more health care provider visits and absences from school and work than any other illness every year. It is estimated that during a 1-year period, people in the U.S. will suffer 1 billion colds. Caused by a virus that inflames the membranes in the lining of the nose and throat, fever, colds can be the result of more than 200 different viruses. However, among all of the cold viruses, the rhinoviruses cause the majority of colds.

5.4.2 *Example 2:*

Clinical Note: NOTEEVENTS-09002.txt

Abbreviation: BP

Sentence from input Clinical Note:

VS - HR 136 RR 60 BP 81/49 61 Temp 100.0 O2 sat 100%

Detected Context Words:

81/49, 61, temp, 100.0, 60, rr, 136, hr, -

Total CUIs for this abbreviation in Metamap with their Meaning:

C2986841 - Binding Potential, C0057191 - bleomycin/cisplatin protocol

C0037623 - Solomon Islands, C0005823 - Blood pressure

C0005824 - Blood Pressure Determination

Disambiguated Accurate CUI: C0005823

Conclusion:

To validate the correction of above result, we verified the abbreviation and context word relation in clinical documents on valid medical websites.

Primary website used for result validation:
https://www.nlm.nih.gov/medlineplus/ency/article/002341.htm

Description related to BP in Clinical Text from Website:

Vital signs reflect essential body functions, including your heartbeat(HR), breathing rate(RR), temperature(Temp), and blood pressure(BP). Your health care provider may watch, measure, or monitor your vital signs to check your level of physical functioning.

6 CONCLUSION

This project implements an unsupervised graph based WSD system for clinical notes using Apache Spark. We used UMLS as the knowledge base to disambiguate medical abbreviations. We observed that despite of generating and loading graph of a huge Knowledge Base, processing time was reduced to a great extent due to usage of spark's in-memory computational features, which helped to reduce the read-write time on disk. Also, the resulting average accuracy for disambiguated abbreviations considering similarity and betweenness centrality measure was closer to 82.5%, which is more compared to other WSD systems, which have been generated using Knowledgebase approach.

We have calibrated the results using clinical notes from MIMIC2 dataset. One clinical note on an average contains 98 ambiguous abbreviations amongst 204 context words. We ran our algorithm on these clinical notes with a window of size five i.e. five context words of abbreviations from left and 5 from right.

However, considering on the performance front, we achieved a performance for data processing around 362 seconds for parsing one clinical note. Here, parsing a clinical note involves steps of preprocessing, graph scanning and calculating the similarity metrics on the sub graph been generated of CUI's corresponding to the abbreviations detected.

This project can be further developed to enhance it for disambiguating all the medical words and do in-memory caching. Along with this we also plan to examine the impact of considering every relations of UMLS and see how the accuracy of disambiguation process can differ.

7 REFERENCES

[1] Agirre, Eneko, and Aitor Soroa.: Personalizing pagerank for word sense disambiguation. In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, 2009, pp. 33- 41

[2] Navigli, Roberto.: Word sense disambiguation: A survey. ACM Computing Surveys (CSUR) 41.2, 2009

[3] Kilgarri, Adam.: Senseval: An exercise in evaluating word sense disambiguation programs. Proc. of the First International Conference on Language Resources and Evaluation,1998

. [4] Navigli, Roberto, and Mirella Lapata.: Graph connectivity measures for unsupervised word sense disambiguation. In Proceedings of the 20th international joint conference on Artifical intelligence (2007) 1683-1688

[5] [Mihalcea, 2005] Rada Mihalcea. Unsupervised large-vocabulary word sense disambiguation with graph-based algorithms for sequence data labeling. In Proceedings of the HLT/EMNLP, pages 411–418, Vancouver, BC, 2005.

[6] Wessam Gad El-Rab, Osmar R. Unsupervised Graph-based Word Sense Disambiguation of Biomedical Documents, pages 1 to 4 HealthCom 2013-11.

[7] Freeman, L.C.: A Set of measures of centrality based on betweenness. Sociometry 40(1), 1977, pp. 35-41

[8] R. Satya: Comparison of supervised and unsupervised learning algorithms for pattern classification. Vol.2, No.2, 2013, IJARAI

[9] MIMIC II: Clinical Database Overview. Internet. www.physionet.org/mimic2. 2 April 2015

[10] Amazon EMR www.aws.com/ElasticMapReduce 1st May 2015

[11] Agirre, Eneko, and Philip Edmonds : Word sense disambiguation Algorithms and Applications. In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, 2007, pp. 33- 41