UNIVERSITÉ DU QUÉBEC

PH.D. THESIS PROPOSAL
PRESENTED TO
UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF PH.D. IN INFORMATION SCIENCE AND TECHNOLOGY

BY
WASSIM EL-KASS

INTEGRATING SEMANTIC WEB AND UNSTRUCTURED INFORMATION PROCESSING ENVIRONMENTS: A VISUAL RULE-BASED APPROACH

GATINEAU, JANUARY 24, 2018

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Stéphane Gagnon, Thesis Supervisor
Département des sciences administratives, Université du Québec en Outaouais

Mr. Michal Iglewski, Thesis Co-Supervisor
Département d'informatique et d'ingénierie, Université du Québec en Outaouais

Mr. Marek Zaremba, President of the Board of Examiners
Département d'informatique et d'ingénierie, Université du Québec en Outaouais

Mr. Mohand Said Allili, Member of the Jury
Département d'informatique et d'ingénierie, Université du Québec en Outaouais

Mr. Stan Matwin, External Evaluator
Faculty of Computer Science, Dalhousie University

Mr. René Witte, External Evaluator
Department of Computer Science & Software Engineering, Concordia University

THIS THESIS PROPOSAL WAS PRENSENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND PUBLIC

JANUARY 27, 2012

AT UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

FOREWORD

This thesis is presented to the University of Québec in Outaouais (UQO) in partial fulfillment of the requirements for the degree of Ph.D. in Information Science and Technology.

With the ever-increasing volume and variety of digital data, most public and private organizations are realizing the importance of their data and other data that can be made available to them. Data can be a big asset and an important competitive advantage that should not be ignored nowadays. This is true only if data is analysed and used properly. Otherwise, data becomes a burden for its storage and maintenance.

A large number of organizations are shifting toward Data-Driven Decision Making (also known as Evidence-based Decision Making). This is, in part, due to major hardware advances in storage capacity and processing speed as well as the progress made in Data Mining and visualization tools that made data analysis available to non-technical domain experts.

Current Data Mining and visualization tools are primarily focused on structured data when the large majority of data being generated today is unstructured. This leaves a huge gap and causes organizations to lose the information and knowledge buried in unstructured data.

Our research was mainly focused on closing the unstructured data gap by providing tools and methods for non-technical domain experts to extract knowledge from unstructured data in an easy and efficient way.

ACKNOWLEDGMENT

I would like to express my deepest and most sincere gratitude to my research supervisor Dr. Stéphane Gagnon and co-supervisor Dr. Michal Iglewski for their continuous guidance, support, motivation, and patience. Their patience and empathy helped me overcome the stress of doing a PhD while maintaining my full-time job as Team Leader, teaching undergraduate courses, and helping my wife with our four kids including newly born twins. Dr. Gagnon's and Dr. Iglewski's availability and willingness to help combined with their wealth of deep knowledge in various domains greatly served me throughout my research.

I also would like to thank my wife Sonia for her support, encouragement, and patience. She often played my role with kids in addition to her role when I was too busy during the last 8 years. I cannot see how my research and this thesis would've been accomplished without her support.

Last but not the least, I would like to thank my family: my mother and father who supported me emotionally and financially since I was born until I became independent and whom I promised 22 years ago to do my PhD, my brothers, and my in-laws for their love and encouragement throughout my research and while writing this thesis.

INTÉGRATION DES ENVIRONNEMENTS DE TRAITEMENTS DU WEB SÉMANTIQUE ET DE L'INFORMATION NON-STRUCTURÉE : UNE APPROCHE VISUELLE FONDÉE SUR DES RÈGLES

Wassim EL-KASS

RESUME

Les informations non-structurées réfèrent principalement au texte, mais aussi à toutes les informations stockées sans une structure de données prédéfinie. Des progrès significatifs ont été réalisés dans le Traitement automatique du langage naturel (TALN), avec des annotations syntaxique et toponymique très fiables utilisant l'étiquetage morpho-syntaxique (Part of Speech (POS) tagging), la segmentation des phrases (Noun Phrase (NP) chunking), et la reconnaissance d'entités nommées (Named-Entity Recognition, NER).

Cependant, l'annotation sémantique reste une tâche difficile, dont la précision et le rappel varient considérablement selon les types de documents et domaines d'application. Tandis que les textes simples tels que des messages électroniques dans un seul domaine peuvent être analysés avec des résultats cohérents, des documents professionnels et scientifiques de taille similaire, comme les nouvelles et les résumés, présentent trop de complexité avec divers vocabulaires et significations ambiguës à travers des phrases et des sections du document. Les principales difficultés restent la relation des concepts entre eux sous forme de graphiques d'annotation, et leur combinaison pour un classement dans une hiérarchie de classes sémantiquement valide et exhaustive.

Dans cette thèse, nous démontrons comment utiliser les technologies du web sémantique, en particulier les ontologies et bases de données de graphes, pour aider à améliorer la qualité (F-score) de ces tâches d'annotation et de classification. Nous intégrons une ontologie formelle avec une plate-forme de TALN standard, la testons sur un corpus de la recherche publique, et rapportons des F-scores supérieurs aux algorithmes d'apprentissage machine antérieurs.

Nous développons et testons une plate-forme innovante, soit une Architecture adaptative à base de règles pour l'extraction de connaissances (Adaptive Rules-Driven Architecture for Knowledge Extraction, ARDAKE). Notre logiciel intègre la norme Architecture de gestion de l'information non-structurée (Unstructured Information Management Architecture, UIMA) avec une base de données graphique standard pour stocker nos ontologies. Nous développons des extensions au langage de règles UIMA Ruta pour invoquer et vérifier les rapports entre classes de l'ontologie. D'autres extensions comprennent le calcul de mesures complémentaire utiles pour intégrer les règles de correspondance (matching) entre mots et classes, soient conditionnelles, statistiques, et basées sur les distances sémantiques. Nous développons également un nouvel algorithme itératif des n-grammes afin de combiner les règles de

correspondance et d’optimiser les F-scores et l’aire sous les courbes de Caractéristique de fonctionnement du récepteur (Receiver Operating Characteristic, ROC). Nous proposons un nouveau style graphique circulaire (pie-chart) pour faciliter la visualisation de l'évaluation de la performance d'annotation. Ces composants sont intégrés dans une interface graphique permettant aux experts du domaine de règles de composer visuellement des ensembles de règles, dans des hiérarchies de complexité variable, de scorer et comparer leur performance relative, et enfin les améliorer en intégrant des sources d'ontologies supplémentaires.

Notre plate-forme est testée sur un cas d'utilisation particulier dans les sciences de la santé : une méthode d'analyse de la littérature médicale selon la population, l’intervention, le contrôle, et les résultats (Population, Intervention, Control, and Outcome, PICO). Nous montrons que notre plate-forme peut efficacement et automatiquement produire des ensembles de règles parcimonieux, avec des F-scores plus élevés sur les classes P et I que les auteurs antérieurs utilisant des algorithmes d'apprentissage machine.

INTEGRATING SEMANTIC WEB AND UNSTRUCTURED INFORMATION PROCESSING ENVIRONMENTS: A VISUAL RULE-BASED APPROACH

Wassim EL-KASS

ABSTRACT

Unstructured information refers primarily to text but also any information stored without a pre-defined data structure. Significant advances have been made in Natural Language Processing (NLP), with reliable syntactic and gazetteer annotations from Part of Speech (POS) tagging, Noun Phrase (NP) chunking, and Named-Entity Recognition (NER).

However, semantic annotation remains a challenging task, with precision and recall varying greatly across document types and application domains. While simple texts such as email messages in a single domain can be analysed with consistent results, professional and scientific documents of similar size, such as news and abstracts, present too much complexity with diverse vocabulary and ambiguous meanings throughout sentences and document sections. Major difficulties remain in accurately relating concepts with one another into annotation graphs, and combining them for further classification across a hierarchy of classes with semantic relevance and completeness.

In this thesis, we demonstrate how to use semantic web technologies, in particular ontologies and graph databases, to help improve the quality (F-score) of such annotation and classification tasks. We integrate a formal ontology with a standard NLP platform, test it on a public research corpus, and report F-scores superior to prior Machine Learning algorithms.

We develop and test an innovative platform, the Adaptive Rules-Driven Architecture for Knowledge Extraction (ARDAKE). Our software integrates the Unstructured Information Management Architecture (UIMA) with a standard graph database to host our ontologies. We develop extensions to the UIMA Ruta rules language to invoke and verify class relationships from the ontology. Other extensions include computing additional text metrics useful in integrating conditional, statistical, and semantic distances for token-class matching. We also develop a new iterative n-grams algorithm to combine matching rules and optimize F-scores and area under the Receiver Operating Characteristic (ROC) curves. We propose a new pie-chart style to facilitate visualization of annotation performance evaluation. These components are integrated within a graphical interface allowing domain experts to visually compose rule sets within hierarchies of varying complexity, score and benchmark their relative performance, and improve them by integrating additional ontology sources.

Our platform is tested on a particular use case in the health sciences: the Population, Intervention, Control, and Outcome (PICO) medical literature analysis methods. We show that our platform can efficiently and automatically produce parsimonious rule sets, with higher F-scores on the P and I classes than prior authors using machine learning algorithms.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

AAE: Aggregate Analysis Engine

AE: Analysis Engine

AI: Artificial Intelligence

ALTA: Australasian Language Technology Association

ANNIE: A Nearly-New Information Extraction

ARDAKE: Adaptive Rules-Driven Architecture for Knowledge Extraction

AUC: Area Under the ROC Curve

BLOB: Binary Large Object

CART: Classification and Regression Trees

CREOLE: Collection of REusable Objects for Language Engineering

CRISP-DM: CRoss-Industry Standard Process for Data Mining

DAML: DARPA Agent Manipulation Language

DM: Data Mining

EBM: Evidence Based Medicine

ETL: Extraction, Transformation, and Loading

FN: False Negatives

FP: False Positives

GATE: General Architecture for Text Engineering

IDF: Inverse Document Frequency

JAPE: Java Annotation Patterns Engine

KE: Knowledge Extraction

KEFUD: Knowledge Extraction From Unstructured Data

LHS: Left Hand Side

MeSH: Medical Subject Headings

ML: Machine Learning

NCBI: National Center for Biotechnology Information

NE: Named Entity

NER: Named Entity Recognition

NICTA: National Information Communications Technology Australia

NLM: National Library of Medicine

NLP: Natural Language Processing

NN: Neural Network

OIL: Ontology Interchange Language

RDF: Resource Description Framework

RDFS: RDF Schema

RHS: Right Hand Side

ROC: Receiver Operating Characteristics

SCA: Service Component Architecture

TN: True Negatives

TP: True Positives

UIMA: Unstructured Information Management Architecture

URI: Universal Resource Identifier

XML: eXtensible Markup Language

CHAPTER 1

Introduction

1.1 Problem Statement

Unstructured data has been growing exponentially for more than a decade with no indication that this trend is going to change in the foreseeable future. According to many studies and statistics, unstructured data represents nearly 80% of all corporate data. Yet, organizations still invest a lot more efforts and resources into data mining to extract knowledge from data structured in relational databases. This is mainly due to the simplicity of extracting knowledge from structured data compared to extracting knowledge from unstructured data. Data mining and visualization tools have evolved over the last twenty years to help, less technical, business users analyse and visualize structured data to gain quick insights and see useful trends.

It is safe to assume that the amount of knowledge embedded in unstructured data is far bigger than what exists in structured data. This can be justified by two properties of unstructured data that are the abundance of unstructured data (medical and non-medical research articles and results, twitter, Facebook, e-commerce feedback, etc.) and the amount of knowledge (implicit and explicit semantic information and relationships) that can be encoded/hidden in even small chunks of unstructured data (short tweets from leaders, politicians, scientists, etc.). These very two properties are unfortunately what make Knowledge Extraction From Unstructured Data (KEFUD) particularly challenging.

Natural languages are extremely expressive, allowing domain experts to express and describe domain knowledge including their findings and results in a way that other domain experts can understand and act on. Humans are good in identifying and relating concepts, emotions, actions, and facts described in naturel languages but are limited when it comes to the number of facts and concepts they can relate and analyze simultaneously. Computers, on the other hand, can efficiently relate and analyze millions of concepts and facts simultaneously, as long as these concepts and facts are formally defined in a way that a computer software can interpret.

Object-oriented programming languages makes it possible for software developers to define concepts and facts as well as the relationships and interactions between different concepts. However, domain expert cannot use programming languages to describe their domain knowledge and knowledge is produced at a much higher rate than what software developers can handle.

What makes things more complex is the large skillset gap and incompatible thinking between developers and domain experts which usually requires involving business analysts to act as middlemen and fill in this gap. Engaging developers and business analysts to capture all domain knowledge in software applications is impractical and has a cost that many businesses cannot financially afford. Another way to make knowledge available for computers to access and act on is to encode this knowledge using a formal language that a software (reasoner) can interpret. A reasoner for a formal language is a software program that has code to handle all keywords (vocabulary) and rules (grammar) of this language. Therefore, the reasoner's complexity increases with the increase of the language complexity and expressive power. Natural languages are extremely high expressive making it impossible to write interpreter (reasoners) for even one language. Many formal languages with different expressive powers were defined for knowledge representation. Tools were also developed to allow creating and managing knowledge encoded using these formal languages.

While this is easier for domain experts than using programming languages, it is still too complex for non-technical people to learn and use formal languages. Data engineers and business analysts would still be required to convert domain knowledge into formal languages that computers can interpret and act on. Again, the amount of existing and new knowledge described in natural languages exceeds by far the capacity to encode all this knowledge in a formal language. Natural Language Processing (NLP) lets domain experts describe their knowledge using natural language and provides tools to process the text in order to identify knowledge or actionable data usable by computer software. NLP helps determining the structure and finding Named Entities (NE) in natural language sentences but is not enough by itself to identify actionable information since it deals mostly with the linguistic aspect of the

text being analysed. NLP can be ineffective and difficult to use in some domains where special terminologies and abbreviations or where languages other than English are used.

1.2 Integrating Text Mining and Semantic Web Technologies

NLP based on machine learning algorithms, and semantic web technologies focused on ontology and reasoning, are two approaches that have found tremendous success. Countless applications have found their way into all sectors of the economy and become generic and well-accepted technologies, such as: email filtering and spam detection, semantic indexing and search, automatic translation and summarization, question-answering systems, user profile associations and recommendations, etc.

Rule-based methods to identify actionable information and extract knowledge directly from unstructured data can be precise and efficient. However, languages used to create these rules are too complex even for advanced software developers. Furthermore, most rule-based languages focus on the linguistic features and partially or completely ignore the semantic dimension of the data.

The increasing demand for more diverse applications of text analytics creates opportunities for improving the performance and reliability of these systems, and make them ever more versatile to address an ever-expanding variety of unstructured contents. For that purpose, combining NLP, semantic, and rules-based approaches present major advantages, and compensate for their respective weaknesses. For example, while bag-of-words techniques remain the most efficient and effective for simple texts (e.g., email), ontology-driven tagging provides for the most exhaustive coverage of potential relations among text patterns, even in higher complexity texts. As well, even if semantic rules can be most formal representation of a text structure, ensuring the right rules are used among an exponential number of combinations still remains a computationally challenging task.

Combining these technologies is in theory feasible, but requires significant development capabilities, and testing before confirmed reliability. Therefore, it is no surprise that such

integrated solutions have remained mostly at experimental level, lacking sufficient performance to find commercial applications.

1.3 Research Objectives

Our thesis addresses these many challenges by focusing first on a key requirement often neglected in this research area: making text analytics engines more user-friendly. With that in mind, we developed a prototype named Adaptive Rules-Driven Architecture for Knowledge Extraction (ARDAKE), providing first an interface for domain experts to seamlessly and collaboratively develop text mining rules. These are then executed, scored, and combined into pipelines or processes using an open source NLP backend, integrated with a graph database and rules language. Once the optimal combination of rules has been found and recommended by the system, it can then be applied to annotate text corpora, reusing as well existing annotators. Our implementation uses the Unstructured Information Management Architecture (UIMA) and the Ruta annotation rules language, to which we added semantic integration extensions.

To demonstrate the effectiveness of our approach, we used a text corpus that was extensively studied using machine learning. We imported the PIBOSO dataset maintained by NICTA in Australia, which had been the focus of a competition at the Australasian Language Technology Conference for several years. All prior authors used various ML tactics to improve sentence classification in a set of 1000 abstracts from the medical literature focused on a single disease (i.e., spinal cord injury). Our results outperformed those obtained by most state-of-the-art ML algorithms used in the ALTA-NICTA PIBOSO contest confirming the simplicity and the effectiveness of our approach as well as the potential of integrating text mining and semantic annotation.

1.4 Structure of the Thesis

This thesis is structured in 10 chapters. In chapter 2, we review the foundations of text mining and classification performance metrics, as well as define core concepts of semantic web technologies and popular platforms, proposing to integrate both approaches to solve performance issues with complex text corpora. Chapter 3 presents the objectives and research

procedure, focusing on the choice of our demonstration corpus and establishing conditions for the successful evaluation of our approach. We propose in chapter 4 an n-gram algorithm to automatically generate annotation rules sets, which are further used within our application along with user generated rules. Our prototype is described at length in chapter 5, explaining especially the operation of our end-user interface and its interaction with the back-end annotation platform and rules language. Chapter 6 defines the application domain of our demonstration, which is sentence classification of medical journal abstracts, and details the many steps to pre-process and prepare the dataset for our annotation platform. Our approach is tested and results are reported in chapters 7, 8, and 9, dealing respectively with data analysis, rules development, and annotation performance scores. Our conclusion in chapter 10 summarizes the contributions and limitations of our thesis, and outlines a future research program in this area.

CHAPTER 2

Literature Review

2.1 Chapter Overview

Computer systems and devices have gone a long way in providing novel services and breaking new records in the last decade. After almost dropping the Artificial Intelligence (AI) dream, to build machines and systems that outsmart humans, at the end of the last century, AI was revived by major advances in increasing the storage capacity and computational power of computers as well as the introduction of new standards, platforms, tools, and libraries to resolve specific issues that undermined the AI success in the past. A major goal of AI is to have systems, agents, or devices that can learn and become smarter over time. Learning is done by acquiring data from different data sources (such as knowledge bases, text, images, videos, etc.), analyzing the data to identify actionable information that can be used to enrich knowledge bases and to make decisions. This requires high capacity and efficient storage devices, standards to represent and exchange knowledge, annotation techniques and tools to identify actionable information in data read from data sources as well as tools that can act on identified information to help enriching knowledge bases, making recommendations, or even take decisions.

High Capacity and efficient storage is important to store and access data from which intelligent systems can learn or analyse to make recommendations and decisions. Storage is not an issue anymore after the introduction of extremely high capacity and high-speed storage devices. Cloud services and Hadoop pushed the storage limit even higher and eliminated the concerns about data size especially for unstructured data. The I/O overhead that used to kill applications' performance in the past was greatly reduced with powerful new technologies and parallel read/write operations.

Since 2000, many efforts were made to create higher expressive knowledge representation semantic languages with acceptable computational complexity. Those efforts resulted first in merging the DARPA Agent Manipulation Language (DAML) and the Ontology Interchange Language (OIL) into one language known as DAML+OIL that became a layer on top of the

existing Resource Description Framework (RDF) and RDF Schema (RDFS) knowledge representation languages. DAMPL+OIL later inspired the creation of the OWL language in 2004. In 2009, OWL2 which is a newer and more expressive version of OWL became a W3C recommendation. OWL2 is supported by editors such as Protégé along with visualization plugins and many plugins for reasoning and semantic inference such as Pellet, RacerPro, FaCT++, and HermiT.

While many studies were done on annotation techniques and tools to identify actionable information in data sources, the results are still far from being satisfactory especially with unstructured data [1]. This demonstrates the need for doing more research in this area in order to improve the performance and efficiency of annotation tools especially with the fast and steady increase of unstructured data. [2] shows the great need for more research in order to advance the state-of-the-art of rule-based information extraction from unstructured data.

According to many statistics, unstructured data, mostly text, account for over 80% of organizations data and is doubling in size every 18 months [3; 4]. Unstructured data often include large amounts of information that could be crucial for businesses and organizations. Most organizations today recognize the importance of unstructured data and consider it an important asset. Organizations that employ effective methods for extracting information from unstructured data and act upon this information are likely to have a big competitive advantage over those who are not benefiting from unstructured data.

Manual information extraction is not an option nowadays given the huge amount of data produced every day and even every second in some domains. The information extraction process should therefore be automated and run by powerful servers with load balancing or in the cloud in order to break down and distribute the heavy processing load.

Natural languages are not suitable for computers processing. Text written in natural languages has lots of implicit, highly contextual, ambiguous, and imprecise information. Natural languages are too large and allow expressing the same information in so many different ways

making it almost impossible to write parsers for natural languages. Natural Language Processing (NLP) helps determining the structure and finding Named Entities (NE) in natural language sentences. NLP can be used to improve the information extraction from natural languages but is not enough by itself since it deals mostly with the linguistic aspect of the text being analysed. For information extraction, chunks of data need to be properly identified and labeled in a way that other systems can interpret and act on. NLP can be ineffective and difficult to use in some domains where special terminologies and abbreviations or where languages other than English are used.

A mandatory step in enabling computer systems extracting and analyzing information from unstructured data is to map this unstructured data or portions of it into elements of a knowledge representation language that computer systems can parse and understand. This mapping is an example of text annotation. The knowledge representation language used for annotation should be expressive enough to allow representing (expressing) as much knowledge as needed. However, increasing the expressiveness also increases the computational complexity of the language and makes it harder to write parsers (reasoners) for it. It is therefore important to maintain a balance between the expressiveness and the complexity of knowledge representation languages used for annotations. Manual annotation is tedious and time consuming. It is possible to do manual annotation on a small scale but, in most cases, automatic annotation is required due to the amount of unstructured data being annotated.

2.2 Research Questions

Knowledge hidden in unstructured data may help finding root causes or even cures for some major illnesses; stop crimes, attacks, and suicide attempts; or give big competitive advantages to organizations. The large amount of crucial knowledge buried inside unstructured data (especially text), the complexity of extracting this knowledge either manually or using currently available tools, and the limited success of current KE tools in extracting the desired knowledge raise a number of research questions.

- What can be done to improve the efficiency of KE tools so that more useful information and less noise is extracted from unstructured data, mostly text, that represents more than 80% of all data?
- Are current KE tools suitable for domain experts who can best describe the kind of the knowledge they are looking for, what properties help identifying the existence or the absence of certain knowledge, and how to use the knowledge when found?
- How can we reduce the complexity of KE tools, make them easier to use, and make them available to more users including non-technical domain experts?
- Is it possible to maintain large knowledge bases using currently available tools or should automatic knowledge base enrichment be used? What role can annotation tools play in automatic knowledge base enrichment?

We will answer these questions and more in the next chapters of this thesis.

2.3 What is an Annotation?

The annotation process is the act of creating annotation objects. Like metadata, an annotation object is a data that describes other data. [5] defines three types of annotations (Informal, Formal, Ontological) as a tuple (a_s, a_p, a_o, a_c) where a_s is the subject of the annotation, a_p is the predicate of the annotation, a_o is the object of the annotation, and a_c is the context in which the annotation was made. The differentiation factor between the three types of annotations is the type of the different elements of the annotation tuple. For informal annotations, a_s, a_p, a_o, and a_c are textual label which makes their automatic interpretation difficult. In formal annotations, a_s, a_p, a_o, and a_c are Universal Resource Identifiers (URIs) allowing for more consistency and automation but remaining limited on reasoning and deriving more knowledge. Full automation and deriving more knowledge beyond what is found in a corpus is possible with semantic annotations where, a_s, a_p, a_o, and a_c are concepts in ontologies as described in section 2.5.

2.4 Natural Language Processing and Text Annotation Performance

Knowledge Extraction/discovery from textual data is an incremental process that starts with identifying small bits and pieces then examines the different relationships amongst them in

order to come up with potential conclusions. Some of these conclusions constitute the input for even bigger conclusions and discoveries. This is in line with the human nature of extracting knowledge from text by reading characters and words in sentences and paragraphs, linking words to concepts and relationships, then matching these concepts and relationships with concepts and relationships acquired in the past in order to come up with conclusions.

A number of text processing, linguistic, and semantic techniques is needed in order to simulate and mimic the human way of extracting knowledge. We first need tools to match and identify specific words or sequence of characters (patterns) in text. Tools are also needed to link patterns to concepts and relationships in predefined knowledge bases in order to infer more knowledge.

Mapping patterns to concepts and relationships is not always simple and straightforward. Depending on the context, the same pattern can be mapped to completely different concepts. NLP constitutes a powerful tool for recognizing named entities and identifying linguistic relationships between different patterns but NLP alone does not give enough semantic information for knowledge extraction. Similarly, semantic annotations are useful in extending the knowledge far beyond what can be found in the text but this cannot be accomplished without proper pattern matching and NLP to determine the context. The following example shows that even simple knowledge extraction tasks cannot be properly accomplished without patterns matching, NLP, and semantic annotations.

Consider the following two sentences:
“Smoking can cause lung cancer” and “Lung cancer can be caused by smoking”

Pattern matching and NLP help creating annotations and understanding the structure of sentences which can lead to more accurate results. As shown in Figure 2.1 below, with NLP, it is possible to recognize the smoking and lung cancer as nouns and to establish the type and the direction of their relationship through the verb “cause”. However, without linking this to

concepts in a knowledgebase, we limit the knowledge that we can discover and the questions we can answer.

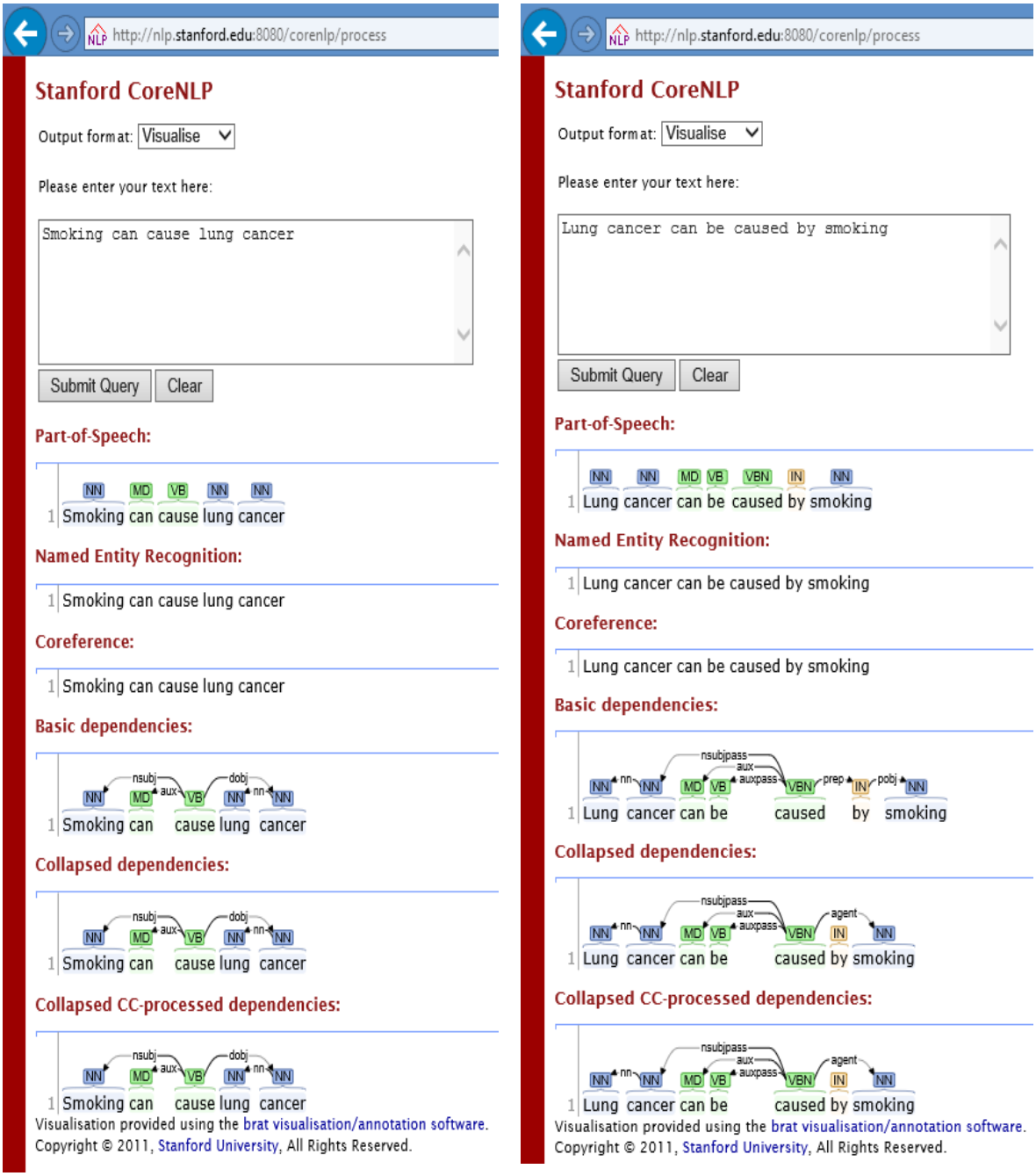


Figure 2.1: NLP chunking

Knowledge and answers for the questions below cannot be obtained from the above two sentences without using a knowledgebase or an ontology. The explanations in parenthesis, beside each question, show how terms from each question can be linked to concepts in the Medical Subject Headings (MeSH) ontology in order to derive more knowledge.

- What diseases are caused by smoking? ("lung cancer" is a subclass of "disease")
- What diseases are caused by tobacco use? ("smoking" is a subclass of "tobacco use")
- What behavior can lead to lung cancers/cancer? ("tobacco use" is a subclass of behavior)
- Does smoking affect the respiratory system? ("lung" is a subclass of "respiratory system")
- Etc…

Knowledge acquiring is different from knowledge extraction retrieval/discovery/uncovering and is more concerned with building knowledge bases by learning new concepts and new relationships. Knowledge acquiring can be theory-based or discovery-based. With Theory-based knowledge acquiring, a number of concepts and their relationships are assumed and tools are used to prove or disprove the theory based on fact data. Discovery-based knowledge acquiring comes more in the form of querying and filtering data then examining the results in order to learn new concepts and relationships.

Given the number of techniques required for knowledge extraction, rich and flexible tools are needed to help end users (especially less technical domain experts) with this complex task. Unfortunately, rich and flexible tools can quickly become unusable due to their complexity and the number of available options that users have to learn and master. We should therefore pay a close attention to keeping knowledge extraction tools as simple and user friendly as possible.

2.5 Semantic Annotations

In order to define semantic annotations, we first need to define what an ontology is. Tom Gruber, who was credited with giving the term ontology a technical definition for computer science, defines the ontology in the context of computer and information sciences as "*An ontology defines a set of representational primitives with which to model a domain of*

knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application" [6]. In section2.7, we present the evolution of ontology languages and discuss their importance in knowledge representation and knowledge extraction.

The semantic annotation is the process of describing data using concepts in ontologies. In other words, semantic annotation is done by linking chunks of data to concepts, instances, and relationships defined in ontologies. Reasoning can then be done on those concepts and relationships to infer even more implicit annotations. The authors of [7] used semantic annotations to introduce and define the semantic web. Nowadays, semantic annotations are perceived as an enabler technology for the semantic web and semantic web services. With semantic annotations, more accurate information retrieval can be achieved in semantic web. The semantic annotation of web service descriptions creates what is known as semantic web services and enable their automatic compositions. This is possible because with semantic annotations, systems are able to interpret the meanings of web services and understand their functionalities assuming that those services are properly annotated [8; 9].

2.6 Performance Evaluation Metrics

Many evaluation methods exist for measuring the performance of machine learning, statistical, and rule-based knowledge extraction models. F-Score, Accuracy, ROC, and AUC are amongst the most widely used methods for comparing and measuring the performance of KE models. Studies show that no single method fits all. While F-Score and Accuracy are considered good for evaluating discrete classifiers (classifiers that class instances as either belong to the class or do not belong to the class), they are criticized for their dependency on the proportional distributions of instances in and outside the class. This is because their formulas take into consideration both positive and negative results at the same time. ROC and its associated AUC method do not suffer from this issue as they are based on False Positive (FP) rate and True Positive (TP) rate that do not mix positive and negative results [10]. When previous studies are

considered as benchmarks and current results are to be compared with the results of those studies, the same evaluation metrics must be used.

2.6.1 Confusion Matrix

A confusion matrix (Table 2.1) is not a performance evaluation metric by itself but it constitutes the basis for many commonly used performance evaluation metrics including Accuracy and F-Score.

Table 2.1: A Confusion Matrix

	Retrieved	Not Retrieved
Relevant	True Positives (TP)	False Negatives (FN)
Irrelevant	False Positives (FP)	True Negatives (TN)

For an IR or KE system or an algorithm that returns the results for a given query, the confusion matrix shows the following four useful evaluation metrics:

- True Positives (TP): The number of relevant results that were correctly identified as relevant.
- False Positives (FP): The number of irrelevant results that were mistakenly identified as relevant.
- True Negatives (TN): The number of irrelevant results that were correctly identified as irrelevant.
- False Negatives (FN): The number of relevant results that were mistakenly identified as irrelevant.

2.6.2 F-Score

The F-Score of a classifier is the harmonic mean of its precision and its recall. The precision is the ratio of the TP results to the total number of retrieved results including the FP results as

in equation (1). The recall is the ratio of the TP results to the number of all relevant results including FN ones as in equation (2). Equation (3) shows the formula for calculating the F-Score of a classifier using its precision and recall.

$$precision = \frac{TP}{TP + FP} \qquad (1)$$

$$recall = \frac{TP}{TP + FN} \qquad (2)$$

$$F - Score = \frac{2 \times precision \times recall}{precision + recall} \qquad (3)$$

An annotation/IR/IE tool or algorithm can be extremely precise but has a low recall. For example, a tool that does not retrieve any irrelevant information but only returns one relevant result out of thousands of relevant results gets 100% precision but a very low recall. Such a tool can be useless for many applications where the priority is to retrieve more relevant results. Similarly, a trivial annotation/IR/IE tool or algorithm that blindly returns all regardless of the query will have a recall of 100% since it is returning all relevant information but its precision will suffer a lot due to the large amount of irrelevant retrieved results. As per the above formulas, the precision is directly proportional with TP and inversely proportional with FP. The recall is directly proportional with TP and inversely proportional with FN. Therefore, the focus should be on increasing the TP and reducing the FP when only the precision matters. To get a better recall, the TP should be increased while decreasing the FN. In order to obtain a better F-Score, one should try to retrieve more relevant results while, at the same time, reducing the number of retrieved irrelevant results.

The F-Score is also known as the F_1-Score or F_1-Measure which is derived from a more generic formula that allows putting more emphasis on either the precision or the recall base on the value of β. The generic formula for F_β-Score is

$$F_\beta - Score = (1+\beta^2)\frac{precision \times recall}{(\beta^2 \times precision) + recall} \qquad (4)$$

2.6.3 Accuracy, Sensitivity, and Specificity

Accuracy is widely used to measure the performance of binary classifications (i.e. whether or not an instance belongs to a given class). It is the ratio of the number of instances properly classified as TP or TN over the total number of tested instances. The Accuracy formula is:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad (5)$$

Other binary classification quality measurements that are widely used in bio and medical applications are the Sensitivity and the Specificity. Like the Accuracy measure, both Sensitivity and Specificity are defined in terms of TP, TN, FP, and FN as shown in the following formulas. Unlike Accuracy that takes into account correct (both positive and negative) classifications, Sensitivity measures the quality of positive classifications only while Specificity measures the quality of negative classifications as shown in equations (6) and (7) [11].

$$Sensitivity = \frac{TP}{TP + FN} \qquad (6)$$

$$Specificity = \frac{TN}{FP + TN} \qquad (7)$$

2.6.4 Receiver Operating Characteristics (ROC)

Receiver Operating Characteristics (ROC) (Figure 2.2) is a graph-based technique for evaluating classifiers and visualizing their performance. ROC graphs are commonly used in medical decision making as well as in the machine learning and data mining. Figure 2.2 shows the curves corresponding to the 3 different classifiers A, B, and C.

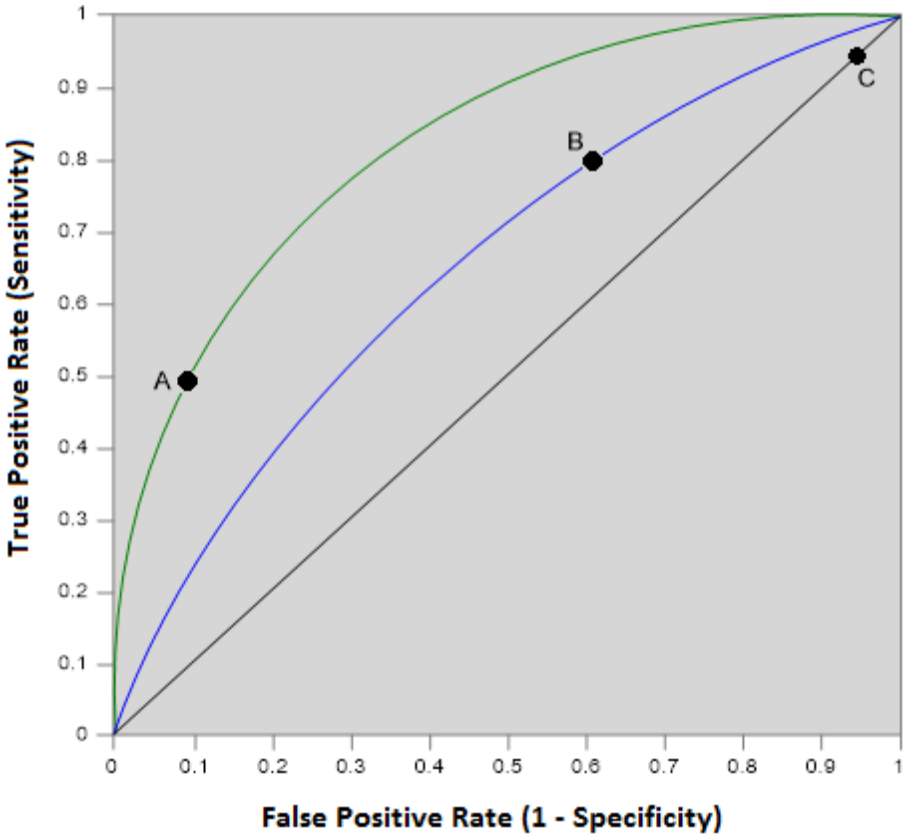


Figure 2.2: The ROC Curve - Modified from [3; 4].

A ROC graph is a two-dimensional graph where the x-axis represents the FP rate (specificity inverse) and the y-axis represents the TP rate (sensitivity). Discrete classifiers strictly classify instances as either belonging or not belonging to a given class. Thus, a discrete classifier produces a pair of (TP rate, FP rate) that maps to a single point on the ROC graph. Non-discrete classifier such as Naïve Bayes and Neural Networks classify instances with a probability or a confidence level. Since instances are not strictly assigned to a given class when using non-discrete classifiers, a threshold must be used to discretize their results in order to map them to points on the ROC graph.

2.6.5 Area Under the ROC Curve (AUC)

As the names implies, the Area Under the ROC Curve (AUC) refers to the area of the zone below the ROC curve in a ROC graph. Figure 2.2 shows that classifier A has a larger AUC than classifiers B and C despite the fact that A has the lowest TP rate or sensitivity. This is because A has a much better specificity than B and C. This means that A has a much lower FP rate. We can also see that C has an AUC of 0.5 (or 50% of the area of the square). This means that predictions produced by C are not any better than random guessing despite the fact that C has a very high TP rate. This can happen for a classifier that classifies all instances as instances of type C1 regardless of their real class. This classifier would have the highest TP rate of 1 as it successfully classifies all instances of C1. However, this same classifier also gets highest FP rate of 1 for failing to classify all instances that do not belong to C1. The ROC curve for (1, 1) coordinate on the ROC graph corresponds to the flat line diagonal of the square that translates into a 0.5 AUC.

[12] proposes the following simple formula for estimating the AUC:

$$A = \frac{S_0 - n_0(n_0 + n_1)/2}{n_0 \times n_1} \qquad (8)$$

where n_0 and n_1 are the number of positive and negative instances respectively and S_0 is the sum of r_i, where r_i is the rank of the i_{th} positive example in the ranked list. [13] shows that AUC is a better measure than accuracy in comparing learning algorithms.

2.7 Integrating Semantic Web Technologies for Annotation

While semantic annotations are primarily used to enable the semantic web and semantic web services, nothing prevents from using them in other general-purpose or domain specific systems. Any application that requires automation, advanced and precise search capabilities, integration, or intelligence may benefit from the use of semantic annotations. The list of such applications is endless and includes decision-support systems, search engines, e-Commerce, healthcare and life sciences, and surveillance applications. Those systems and applications can benefit, not only from semantic annotations, but also from other semantic web technologies such as semantic rules, reasoning, semantic query languages like SPARQL, triple stores, and

so on. The following sub-sections show the evolution of modern knowledge representation languages that were created to enable and support the semantic web technologies but can play a key role in semantic annotation and knowledge extraction.

2.7.1 XML and XML schemas

XML was a big step forward toward storing data in both human and machine-readable format. Data encoded in XML documents is machine readable (i.e. can be separated into different data elements) but not machine-interpretable unless the structure and the definitions of tags are considered in the application logic. This is because different people can use different XML tags and different data types and formats to store the same data in XML formats. XML schemas describe the structure and the data types of XML elements making it possible to implement generic logic to interpret and validate any XML document that adhere to the defined schema. This is possible because the generic logic is implemented based on the predefined schema and not based on individual XML documents. XML schemas allow defining what data elements and attributes can be used in respective XML documents. An XML schema also defines the structure of data elements (nodes) in corresponding XML documents where child nodes are contained in their parent nodes.

2.7.2 RDF

Besides defining the types and the parent-child elements relationship, XML schemas do not define any semantic or standard way to characterize data elements or their relationships. In plain XML, nodes <Description>, <xyz>, and <Disease> are semantically equivalent because they are just nodes. When dealing with plain XML, the semantic of various nodes is defined at the application level. As such, there is no standard way, using plain XML or even using an XML schema, to denote a relationship between two concepts (ex.: Smoking causes Lung Cancer). Plain XML and XML schemas are therefore not suitable for semantic annotations and knowledge representation which requires describing concepts and relationships between concepts in a formal and standard way. RDF remediates this problem by defining the semantic

of some elements and attributes making it possible to describe resources (especially web resources) and to express the relationships between different resources in a standard way as shown in the code snippet below.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE rdf:RDF[
        <!ENTITY rdf 'http://www.w3.org/TR/rdf-syntax/'>
        <!ENTITY rdfs 'http://www.w3.org/TR/rdf-schema/'>
        <!ENTITY rel 'http://www.mydomain.org/relations/'>
        <!ENTITY nci 'http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#'>
        <!ENTITY bio 'http://purl.bioontology.org/ontology/PDQ/'>
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:rel="&rel;" xmlns:nci="&nci;"
        xmlns:bio="&bio;">
        <rdf:Description rdf:about="&nci;C17934">
                <rdfs:label>Tabaco Smoking</rdfs:label>
                <rdfs:subClassOf rdf:resource="&nci;C20134"/>
                <rel:causes rdf:resource="&bio;CDR0000040209"/>
        </rdf:Description>
</rdf:RDF>
```

Code Snippet 1

Any RDF enabled system understands that the above code snippet is to describe the resource at URI “http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C17934” using resources “Tabaco Smoking”, “http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C20134”, and “http://purl.bioontology.org/ontology/PDQ/CDR0000040209” with the ‘label’, ‘subClassOf’, and ‘causes’ relationships defined at the rdfs, rdfs, and rel namespaces respectively. The RDF enabled system can also follow related resources and learn more about their properties and relationships. This is how RDF enables linked data and the semantic web. The RDF semantic model is based on triplets that represent data relationships in the form of <s, p, o> where s is the subject, p is the predicate, and o is the object.

2.7.3 RDF serialization

The RDF/XML formant in code snippet Code Snippet 1 was the first standard RDF serialization format but the wide adoption of RDF led to the support of many other RDF serialization formats. Table 2.2 shows today's most common RDF serialization formats.

Table 2.2 : Most Common RDF Serialization Formats

RDF serialization format	Description
Turtle	a compact, human-friendly format
N-Triples	a simple, easy-to-parse, line-based format that is not as compact as Turtle
N-Quads	a superset of N-Triples, for serializing multiple RDF graphs
JSON-LD	a JSON-based serialization mostly used in REST lightweight web services
N3 or Notation3	a non-standard serialization that is similar to Turtle, but has some additional features, such as the ability to define inference rules.

2.7.4 RDF Graphs

A set of RDF triples can be represented as a directed labeled graph where the nodes represents the subjects and the objects of the RDF triples and where the arcs represent the predicates linking subjects to objects. Figure 2.3 shows the RDF graph for RDF Code Snippet 1.

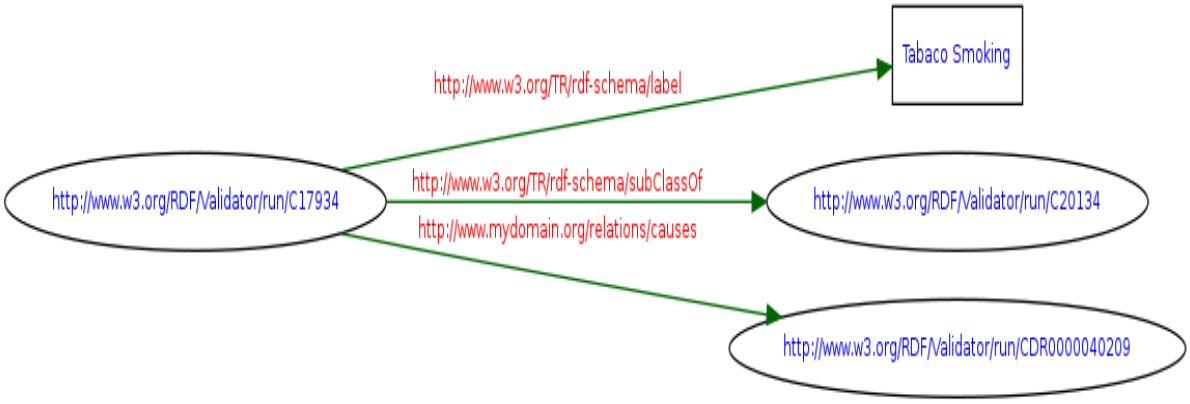


Figure 2.3: An RDF Graph

2.7.5 RDFS

RDF provides a mechanism to describe domain-neutral metadata but does not define the semantics of any domain application [14]. While RDF-enabled systems are able to detect relationships between resources, those systems do not understand the semantics (meanings) of the relationships and are therefore unable to act upon them or exploit them to derive more knowledge or take further actions. In code snippet Code Snippet 1, the subClassOf relationship does not mean anything, beyond being just a link between two resources, to an RDF-enabled system. This RDF limitation led to the creation of many RDF extensions with richer semantics and more expressiveness power such as the RDF Schema (RDFS) and OWL.

RDFS introduced a number of elements including (class, subClassOf, property, subPropertyOf, domain, label, range, type) that can be used to define domain-specific classes (concepts) and their properties. An RDFS enabled system reading code snippet Code Snippet 1 concludes that the resource being described is labeled “Tabaco Smoking” and is a subclass of another concept labeled “Smoking Behavior” which is in turn a subclass of a concept labeled “Personal Behavior” and so on. This means that any property or logic that applies to “Smoking Behavior” or “Personal Behavior” also applies to “Tabaco Smoking” as this latter is a subset of the previous two. From an information or knowledge extraction perspective, it means that results

matching “Tabaco Smoking” should be returned when searching for “Personal Behavior” which is not possible with pure syntactical pattern matching.

Despite its richness and expressive power, RDFS remains far from being sufficient to describe and represent domain knowledge. Many basic constructs that are fundamental for knowledge representation and extraction cannot be expressed in RDFS. This includes class conjunction, disjunction, cardinality restrictions as well as properties constraints such as uniqueness, inversion, and transitivity. OWL and its extension OWL2 were therefore introduced to overcome the shortcomings of RDFS [15].

2.7.6 OWL and OWL2

OWL and OWL2 were created in response to the increasing demand and need for more expressive languages allowing better and richer knowledge representation. Unfortunately, the expressiveness has a costly price. The more expressive a language is, the higher is its computational complexity. Some operators, such as the negation operator, are known for their computation complexity that can be exponential. Writing parsers or reasoners is also harder for the more expressive languages.

To maintain a balance between the expressiveness and the complexity, OWL 2 was broken down into 3 different profiles or sublanguages where each profile trades some expressive power for the efficiency of reasoning and is suitable for specific application scenarios.
OWL 2 EL: Mainly intended for applications employing ontologies with very large numbers of properties and/or classes. OWL 2 EL is efficient for applications that require the recognition of structurally complex objects. It has a reasoning complexity of PTIME-complete.
OWL 2 QL: Intended for applications that use very large volumes of instance data, and where reasoning is mostly based on query answering. OWL 2 QL is suitable for representing database schemas and has a reasoning complexity of NLogSpace-complete.
OWL 2 RL: is designed for applications that require scalable reasoning while maintaining the highest possible expressiveness power. Its reasoning complexity is PTIME-complete.

2.7.7 Semantic Reasoners

A semantic reasoner or simply reasoner is an application that can interpret a set of facts, typically defined in an ontology, and infer logical consequences that are implicit and may be hard to identify manually especially in large ontologies. The standardization of OWL led to the creation of a number of OWL and OWL2 reasoners. Pellet, FacT++, RACER, and HermiT are among the best available OWL reasoners.

Given enough facts, a reasoner may reveal crucial implicit knowledge that are otherwise hard to discover. For example, if a fact indicates that depression leads to smoking for individuals with certain characteristics and another fact indicates that smoking causes lung cancer, a reasoner can conclude that individuals with certain characteristics are more at risk to have lung cancer if they have a depression. This looks like a trivial conclusion to make but it may be impossible to recognize when looking into a large number of facts.

2.7.8 Triple stores

Triple stores are special graph databases for the storage and retrieval of RDF triples. They can store billions of triples while maintaining extremely high read/write performance. An SQL-like query language called SPARQL is used to query data in triple stores.

There are many commercial and free triple stores available today. Oracle Spatial and Graph with Oracle Database 12c is the most powerful one and can load, infer from, and query up to 1.08 Trillion triples. AllegroGraph by Franz is another triple store that can handle up to 1 Trillion triples. Stardog 2.1 by Clark & Parsia can work with 50 Billion triples on a 32 cores server with 256GB of RAM reaching load speeds of 500k triples/sec for 1 Billion triples and more than 300k triples/sec for 20 Billion triples. Stardog offers a free community version that does not require any licence fees and supports 10 databases with up to 25 Million triples in each database. Google Knowledge Graph (previously FreeBase) is another interesting triple store that can handle billions of triples.

More details on the performance numbers and machines used for testing different triple stores can be found at: http://www.w3.org/wiki/LargeTripleStores.

2.8 Platforms and Standards for Annotation Rules and Pipelines

Efficient knowledge representation and sharing is not possible without defining common standards to describe, serialize, and communicate the knowledge. A knowledge representation standard must define how to describe simple and complex concepts as well as the relationships between different concepts. When data is unstructured, it is hard for software systems to even detect the boundaries and identify basic concepts. Delimiters and tags can be used to determine the start and end of different data elements in unstructured documents creating a certain structure in those documents. This makes unstructured documents more machine-readable.

2.8.1 GATE and GATE-based Platforms

The General Architecture for Text Engineering (GATE, ([16])) was first released in 1996 at the University of Sheffield. GATE started as a Software Architecture for Language Engineering and evolved into a family of language processing products and tools. The GATE product family includes:

1. GATE Developer: an open source java-based integrated development environment for language processing bundled with many Information Extraction and NLP plugins under what is known as the Collection of REusable Objects for Language Engineering (CREOLE). GATE developers can extend the CREOLE library with their own plugins and use these plugins in their text processing workflows. The core CREOLE library includes Information Extraction and NLP plugins such as ANNIE, LingPipe, OpenNLP, and many NLP parsers and taggers.
2. GATE Teamware: a web-based management platform for collaborative annotation and curation. It enables the coordination of efforts for a distributed team while monitoring the progress and results in real-time. GATE Teamware allows viewing, and editing text annotations remotely through a web browser. Users can load document collections, create

project templates, create projects based on existing templates, assign projects and roles to specific users, monitor the progress of various projects, and generate project status and progress reports.

3. GATE Mímir: a multi-paradigm information management index and repository that can be used to index and search over text, annotations, semantic schemas (ontologies), and semantic meta-data (instance data). It allows queries that mix full-text, structural, linguistic and semantic queries and that can scale to gigabytes of text. Mímir provides indexing infrastructure for annotated GATE documents. Users can submit documents to a Mímir server for indexing. They can then run queries against the set of indexed documents.
4. GATE JAPE: the Java Annotation Patterns Engine (JAPE) is a finite state transducer that operates over annotations based on rules and regular expressions. The GATE framework allows running multiple JAPE transducers in sequence. This chaining allows later JAPE transducers to use the output of earlier JAPE transducers in order to build more and more complex annotations and incorporating more of the context (semantics) of the document into the new annotations. The JAPE grammar consists of a set of sentences, each of which containing a set of pattern/action rules. A JAPE rule has two sides: Left Hand Side (LHS) and Right Hand Side (RHS). The LHS specifies the identified annotation pattern and may contain regular expression operators (e.g. *, ?, +). The Right RHS indicates the action to be taken on the matched pattern and consists of annotation manipulation statements.

2.8.2 UIMA and UIMA-based Platforms

Unstructured Information Management Architecture (UIMA, [17]) is a component architecture and software framework implementation for the analysis of unstructured content like text, video and audio data. For example, UIMA can be used to process business news and identify entities, such as companies, owners, etc., as well as relationships among entities, such as merger or acquisition.

The core of UIMA is the Analysis Engine (AE), a collection of Analysis Components (or Annotators). A primitive AE contains only one Analysis Component, and the UIMA Software

Development Kit makes it easy to statically combine AEs to form an Aggregate Analysis Engine (AAE). This is done through an XML descriptor where no code is required. For more complex (non-sequential) annotation workflows, flow controllers can be inserted into an AAE to determine the order in which the components of the aggregate are invoked. The UIMA framework runs AEs by looping through their Analysis Components and calling their methods in a well-defined order.

While creating primitive annotators is relatively simple, creating aggregate annotators that require more than a simple sequential execution of their component annotators involves many additional steps. These steps include writing and compiling the code that defines the actual execution flow. This makes aggregate annotators harder to implement and maintain, and less adaptable to changes because changing the execution flow requires code changing and recompiling.

To further increase the adoption of UIMA and the reusability of annotators, it must be fairly easy to create and modify aggregate annotators even when the execution flow of their component annotators is complex. This can be achieved by automating the creation of annotators' execution flows based on domain-specific rules [18].

2.8.2.1 Ruta (Formally TextMarker)

Built upon UIMA, Ruta is an open source rule-based language for information extraction and text processing tasks [19]. Knowledge engineers can create scripts of Ruta rules and execute them to identify desired text patterns and create annotations.

Rules are built using a specialized representation language for knowledge formalization, with the possibility of creating new annotation types and integrating them into a taxonomy. They can either be extracted directly from unstructured sources or coded in the scripting rules language. Each rule can be defined within a rule set or type, and associated with specific patterns and actions. Ruta provides a rich set of patterns, conditions, and actions that can be used to create annotation rules. What makes Ruta more interesting is its extension interface

that allows developers to enrich the language by adding more patterns, conditions, and actions. The advantages of Ruta are undermined by its complex syntax as it is the case with all information extraction rules. Providing a visual version of Ruta that is easy to use by non-technical domain expert would unleash the power of Ruta and increase its adoption.

2.8.2.2 U-Compare and Taverna

U-Compare is an integrated text mining/natural language processing system based on the UIMA Framework. U-Compare offers a GUI for easy drag-and-drop workflow (UIMA component descriptor) creation, comparison by U-Compare parallel component, evaluation, statistics, and visualizations. U-Compare workflows have the same limitations that UIMA aggregate components have; that is, only linear fixed flow and language dependent flow are available for use. For more complex execution flow, code is required.

The U-Compare integration within the generic workflow system Taverna offers more capabilities by enabling more flexible and richer annotations and text mining workflows [20; 21]. However, those workflows are static Taverna workflows that cannot be reused by any other UIMA compatible system.

2.8.2.3 IBM LanguageWare

IBM LanguageWare allows extending UIMA annotations using a rules engine. The LanguageWare rules engine supports three kinds of matching rules:

- Break Rules: specify how documents are split into lexical components such as paragraphs, sentences and tokens. A token can be anything from a word to a punctuation symbol, a number, a currency, etc.
- Character Rules: these are character expressions used to match desired sequences of characters such as postal codes, telephone numbers, email addresses, and so on. The rules engine creates annotations when text sequences are found to match the character expression.
- Parsing Rules: use textual patterns from custom dictionary entries and different parts-of-speech, in addition to some previously created annotations.

LanguageWare allows adding features to the newly created annotation type by dragging and dropping features from existing annotations. However, LanguageWare works only on already annotated documents (for reusing existing annotators) and does not support features' computations (only drag-drop of features is supported) nor does it allow Boolean conditions for creating new annotations [22].

2.9 Choosing the Right Annotation Platform

With the abundance of available platforms and tools, it was hard to choose the right one to work with. The choice was particularly difficult between GATE and UIMA as they both offer rich and advanced annotation tools. After working with both GATE and UIMA, we came to the conclusion that UIMA could better serve our research and is more user friendly. A survey done in 2012 (Table 2.3) [23] showed that, except for some interchangeability, UIMA supported all the GATE features. The same survey shows that UIMA supports many key features that are not supported by GATE or the other platforms considered by the study.

Table 2.3: Comparison of NLP architectures from [23]

Comparison of NLP architectures: "+" fully supported, "0" partially supported, "-" not supported.

	Tipster	Gate	Ellogon	HoG	Uima
Stand-off annotations	+	+	+	+	+
Typed annotations	0	+	+	+	+
Annotation Type inheritance	-	-	-	-	+
Processing Resource inheritance	-	+	-	-	0
Processing Resource interchangeability	0	+	+	+	+
Language Resource interchangeability	-	0	-	-	-
Access Structure interchangeability	-	0	-	-	-
Parameter Management	-	+	+	0	+
Analysis Awareness	-	-	-	-	0
Resource Management	-	-	-	-	0
Workflow Management	-	0	0	0	+
Parallelizable	-	-	-	-	+
Distributable	-	-	-	-	+
Tool-Box	0	+	+	-	+

A particularly attractive feature the UIMA rule language (Ruta) is its compactness compared to the GATE rule language (Jape) and other rule languages as clearly demonstrated in [24].

2.10 Research Challenges in Knowledge Extraction

Knowledge Extraction (KE) can be either rule-based or based on statistical or Machine Learning (ML) Data Mining (DM) methods and algorithms. A hybrid approach using both rules and ML for KE is also possible. Each approach has its advantages and disadvantages as explained in the following paragraphs.

Rule-based KE is based on a set of rules that are mostly created and maintained manually using a programming or a rule language. This is the main reason behind criticizing rule-based KE as labor intensive and requiring specialized skills [2; 24; 25]. Resources with various skillsets are required to create KE extraction rules. Domain experts are the best to describe most KE rules since they know their domain rules and exceptions more than anyone else. However, most domain experts do not have the necessary programming skills to write even simple KE rules. Therefore, programmers are needed to create the rules and often business analyst are required to facilitate communications between domain experts and programmers.

Rule-based KE is also criticized for being inefficient when analysing erroneous or incomplete data. Data cleansing to fix erroneous and incomplete data has been extensively studied in the last two decades and will not be covered in this thesis [26; 27; 28; 29].

Despite the inconveniences, rule-based KE has some significant advantages such as debugging and explanation ability. In addition, most rules can be created without relying on training data. Rule-based KE is crucial for industry practitioners but more research is needed to advance its state-of-the-art [2].

ML algorithms can be either supervised or unsupervised. A supervised ML algorithm is trained using a training data set where the results (output variable) have been pre-established. A typical example of a training set is a list of past observations for temperature, humidity, air pressure and other measurements that could indicate rain forecasts and an output variable indicating whether or not it rained the next day. A supervised algorithm learns from the training data by looking at variations and combinations in the input data that lead to a decision matching the output variable in the training data. For the weather forecast example, the algorithm may

attempt to identify the patterns or the combinations of values or ranges that lead to a decision on whether or not it will rain the next day.

Unlike supervised algorithms, unsupervised ML algorithms do not need the output variable in the training set but most of them still require a training set to build their models. They often look for similarities amongst the data being analysed and try to group similar elements together. Many unsupervised ML algorithms exist today to serve different purposes such as Market-Basket and Cluster analysis. Unsupervised algorithms are usually generic and don't have any prior knowledge about the domain or the data being analysed.

Depending on the algorithm, the output of the training exercise of a supervised ML algorithm is a prediction model in the form of a set of rules, a decision tree, a Bayesian or a Neural Network (NN) where the structure and the content of the model are mainly shaped by the underlying training data. A good training set should accurately represent the test data or the real data to be analysed by the prediction model generated by the ML algorithm.

ML KE algorithms are adaptable to various domains and eliminate the need to manually create and maintain KE rules. Nevertheless, domain experts must dedicate significant time and efforts to create and maintain training sets when such sets do not exist. Furthermore, using these algorithms to create or retrain KE model requires ML expertise.

Our research is mainly concerned with rule-based KEFUD. Table 2.4 summarizes the rule-based KEFUD challenges that we intend to address or minimize as part of our research.

Table 2.4: Rule-based KEFUD Challenges

Challenge	**Description**
Creating and maintaining KEFUD rules	Building a KEFUD solution usually requires the creation of very complex pattern matching models. When rule-based KEFUD is used, a large variety of rules requiring different unrelated skillsets is needed to match different kinds of patterns. Additionally, most existing rule-

	based KEFUD solutions require defining rules in a script-like format that requires advanced programming skills. This makes it near to impossible for a single person to define all required rules even for a relatively small KEFUD project. Maintaining these rules and adapting them to changing business requirements is even a bigger challenge.
Creating semantic rules based on ontologies	A well-defined domain ontology constitutes a substantial part of the Knowledge Base in any domain. Medical ontologies, for example, contain the definitions of different illnesses and the relationships between them. These can be linked with pharmaceutical ontologies containing definitions for different kinds of medications, the illnesses they treat, their possible side effects and so on. Domain experts should be, or can easily become, familiar with their domain ontologies as they contain knowledge that they already know or can quickly learn. The big challenge for domain experts is to learn how to use the rich knowledge embedded in their domain ontologies to extract, possibly large amount of, valuable knowledge buried under huge piles of unstructured (typically textual) data. On the other hand, it is usually too hard for, even highly skilled, non-domain experts to grasp the definitions and knowledge embedded in domain ontologies in order to use them for KE. Big opportunities can be lost, resulting in poor KE performance, if the concepts and knowledge of domain ontologies are not properly utilized.
Determining the right set of n-grams to use for KEFUD rules	As we show in details in CHAPTER 4, n-grams play a key role in KEFUD when used properly. Many n-gram aspects such as the length, frequency, and co-existence with other n-grams should be considered when using n-grams for KEFUD. The challenge here is to decide what n-grams to use when building KEFUD models or rules. Many algorithms exist today for calculating the n-grams to include in their KEFUD models. Most of these algorithms are based on the frequency count of n-grams in a training set. They mainly rely

	on the positive correlation between n-grams and the pattern in question without taking full advantage of the negative and collective n-grams correlation with the patterns of interest to improve the quality of their pattern matching results.
Finding the best combination of KEFUD rules	Creating and measuring the performance of elementary inclusion and exclusion pattern matching rules for KEFUD can be hard and time consuming. However, it is usually much harder and way more time consuming to find the right combination of rules to use for KEFUD. The right combination of rules is the combination that produces the maximum KEFUD performance. Evaluating one combination can take hours based on the size of the training set and the complexity of rules. With only 10 elementary rules, one may need many days to test over 1000 combinations in order to find the best one.
Evaluating and optimizing KEFUD rules	KEFUD rules are often evaluated and optimized to improve their outcome. Optimization can be done by tightening the condition of the rule to improve its precision or relaxing it to improve its recall. When advanced metrics such as F-Score are used to measure the performance of a rule, the rule's precision and recall are blended together to produce the final score. This makes it hard for rules designers to know what adjustments they need to apply to their rules.

2.11 Chapter Summary

We reviewed the history and the evolution of KE with a special focus on unstructured data, specifically text, that represents more than 80% of all internet and corporate data and is doubling in size every 18 months according to many statistics. We also described the major challenges encountered with Knowledge Extraction From Unstructured Data (KEFUD) and presented our research questions that we answer throughout the remaining chapters of this thesis.

While many studies were done on annotation techniques and tools to identify actionable information in data sources, the results are still far from being satisfactory especially with unstructured data. Studies show the great need for doing more research in this area in order to improve the performance and efficiency of annotation tools especially with the fast increase of unstructured data sources. A big part of this chapter was to discuss semantic annotations as we believe in the important role they can play in improving the quality of KEFUD. The quality of KEFUD and KE in general is measured using a number of standard performance evaluation metrics that we also defined in this chapter. We finished the chapter with a comparison between the most widely adopted NLP architectures and annotation platforms that shows the many features of UIMA that no other platform currently has.

The rest of this thesis is structured as follow: in chapter 3, we define our research objectives and methodology; in chapter 4 we show the importance of n-grams for KEFUD and introduce a novel method for creating n-gram decision trees that can be used to automatically generate rules for better quality KEFUD; chapter 5 presents the architecture and main components of our prototype called ARDAKE (Adaptive Rule-Driven Architecture for Knowledge Extraction) as well as our extensions to the UIMA Ruta language to include ontology-based semantic rules and other useful statistical and textual rules; chapter 6 covers the challenges of finding and preprocessing a good Corpus for a KEFUD project; in chapter 7, we demonstrate our Corpus data analysis work and show the importance of visual analytics at this stage; chapter 8 presents the rules we developed using ARDAKE to identify sentences containing Population and Intervention patterns in the NICTA-PIBOSO corpus; and evaluate our rules and results and compare them with those obtained by the state-of-the art algorithms in chapter 9. Finally, chapter 10 summarizes the contributions and limitations of our thesis, and outlines a future research program in this area.

CHAPTER 3

Objectives and Methodology

3.1 Chapter Overview

Complex tasks and projects require simple, yet, powerful tools. Large and long projects require methodologies to coordinate and guide the efforts through the various stages and keep the focus on the things to accomplish at each stage. Knowledge Extraction From Unstructured Data (KEFUD) projects are often long and complex and therefore require good methodologies and a rich set of powerful, user friendly, tools to guide miners and help them with their most complex tasks. A key challenge with unstructured data is the identification of semantic concepts and relationships.

In this chapter, we present our research objectives that are primarily aimed at simplifying and improving the performance of rule-based KEFUD projects. Our efforts are to serve two main purposes: Identify the most complex tasks in a rule-based KEFUD project then design and build a set of powerful, easy to use, tools to simplify these tasks with a special focus on the semantic aspects; a methodology specifically designed for rule-based KEFUD projects.

3.2 Research Objectives

Text mining or KE from text has made a long way but has not matured enough to fulfill its goals and cover the needs of the numerous applications that rely on it such as question answering, text understanding/summarization, translation, and knowledge discovery. As discussed in 2.10, current KE techniques, especially the rule-based ones, are quite complex, making it impossible for non-technical domain experts to get the full benefits of knowledge extraction. They require the collaboration of people with many different skills including analysts, designers, developers, statisticians, testers, and more, in addition to domain experts.

Our research objectives are driven by the rule-based KEFUD challenges described in Table 2.4. Table 3.1 lists our research objectives and the challenge targeted by each objective.

Table 3.1: List of Research Objectives

Research Objective	**Targeted Challenge**
Simplify the creation and maintenance of KEFUD rules.	Creating and maintaining KEFUD rules
Make it easy for users, particularly domain experts, to rely on ontology concepts and relationships while creating their KEFUD rules.	Creating semantic rules based on ontologies
Explore the full potential of n-grams including their positive, negative, and collective correlation with the patterns of interest in order to get a better KEFUD performance.	Determining the right set of n-grams to use for KEFUD rules
Find a simple, accurate, and efficient way to identify the best combination of KEFUD rules.	Finding the best combination of KEFUD rules
Make it trivial for rule designers to correct a failing rule in either matching true positive results or avoiding false positive results.	Evaluating and optimizing KEFUD rules

Our overall goal is to make rule-based KEFUD projects simpler, requiring less time and resources, while maintaining their efficiency. We intend to do this through the definition of a clear and complete methodology specifically designed for rule-based KEFUD and the creation of efficient, easy to use, tools to help rule-based KEFUD miners with their most complex tasks. We will rely on visualization to simplify many complex tasks and to eliminate, or reduce, the need for some resources such as programmers and business analysts. To help maintain efficiency, we will enable non-technical domain users to create and maintain various type of rules, including semantic rules, in a simple visual and uniform way. We will also provide tools to automatically generate rules based on common tokens called n-grams. Other tools will help finding the best rules combination that produce the highest classification F-Score.

In order to evaluate our work and to prove that our objectives are met, we will use our methodology and tools to show how easy it is to create KEFUD rules, without any programming, for classifying sentences in the NICTA-PIBOSO corpus and then compare our classification results with those obtained by the state-of-the-art ML algorithms.

Our tools are not limited to sentence classification. Other types of unstructured information processing like Named Entity Recognition (NER), sentiment analysis, link analysis, and language detection can be implemented using specialized rules but this is outside the scope of this thesis. Our tools can be extended to extract information and knowledge from non-textual unstructured data like images, audio, and video. This is possible because our tools are based on UIMA which is a standard framework from processing all types of unstructured data.

Although, it is possible to adopt a hybrid approach by using the output of some ML algorithms as input to our KE rules, we don't consider this as an objective as the integration between ML and rule-based KE is already done in most hybrid KE solutions.

3.3 Research Procedure

We started by studying the latest research on SOA and SaaS and whether or not the current development methodologies are suitable for building SOA and SaaS solutions. This led us to studying the automation of web services composition and how it can be achieved using annotations and semantic web services.

To understand semantic web services, it was necessary to learn about Ontologies, semantic annotations, and reasoning. While studying domain ontologies, we came across the challenging tasks of creating and maintaining them. Semantic annotation is a key enabler but also one of the biggest challenges for semantic web services. As we further investigated the semantic annotation challenge, we realized that annotations play even a bigger role and present a bigger challenge for Knowledge Extraction from unstructured data.

Creating the right annotations in textual data and combining annotations to build more interesting knowledge became our main focus. We studied and compared the state-of-the-art text analysis frameworks and tools including UIMA, LanguageWare, GATE, GATE TeamWare, UCompare with Taverna, Orange4WS, and more. We studied the pros and cons of each framework/tool and came to the conclusion that UIMA was the most powerful and promising due to its flexibility and richness. UIMA makes it easy to define Analysis Engines and provide a framework to combine and run Analysis Engines.

A major limitation for UIMA was the need to write Java or C++ code in order to create Analysis Engines or to conditionally run Analysis Engines based on the results of previous ones or other conditions. Another limitation of UIMA is the lack of integration with Ontologies to create semantic annotations.

We started remediating the UIMA limitations by defining a new architecture that we called Adaptive Rule-Driven Architecture for Knowledge Extraction (ARDAKE). ARDAKE's main goal is to simplify and improve KE from unstructured text and make it available to domain (subject matter) experts who are not familiar with programming and scripting languages such as UIMA Ruta and GATE JAPE. ARDAKE leverages Ruta by adding semantic, linguistic, and statistical rules extensions and allowing users to create and combine these rules in a uniform and visual way.

To further simplify and improve KEFUD, we defined a highly visual process for creating and combining KE rules. Our process was inspired from the CRoss-Industry Standard Process for Data Mining (CRISP-DM) [30] that is the most widely used data mining methodology [31].

Figure 3.1 presents our KEFUD process and how it maps to CRISP-DM. Along with tools allowing visual rules creation for KEFUD, this process greatly simplifies and shorten the duration of KEFUD projects while helping getting the best results.

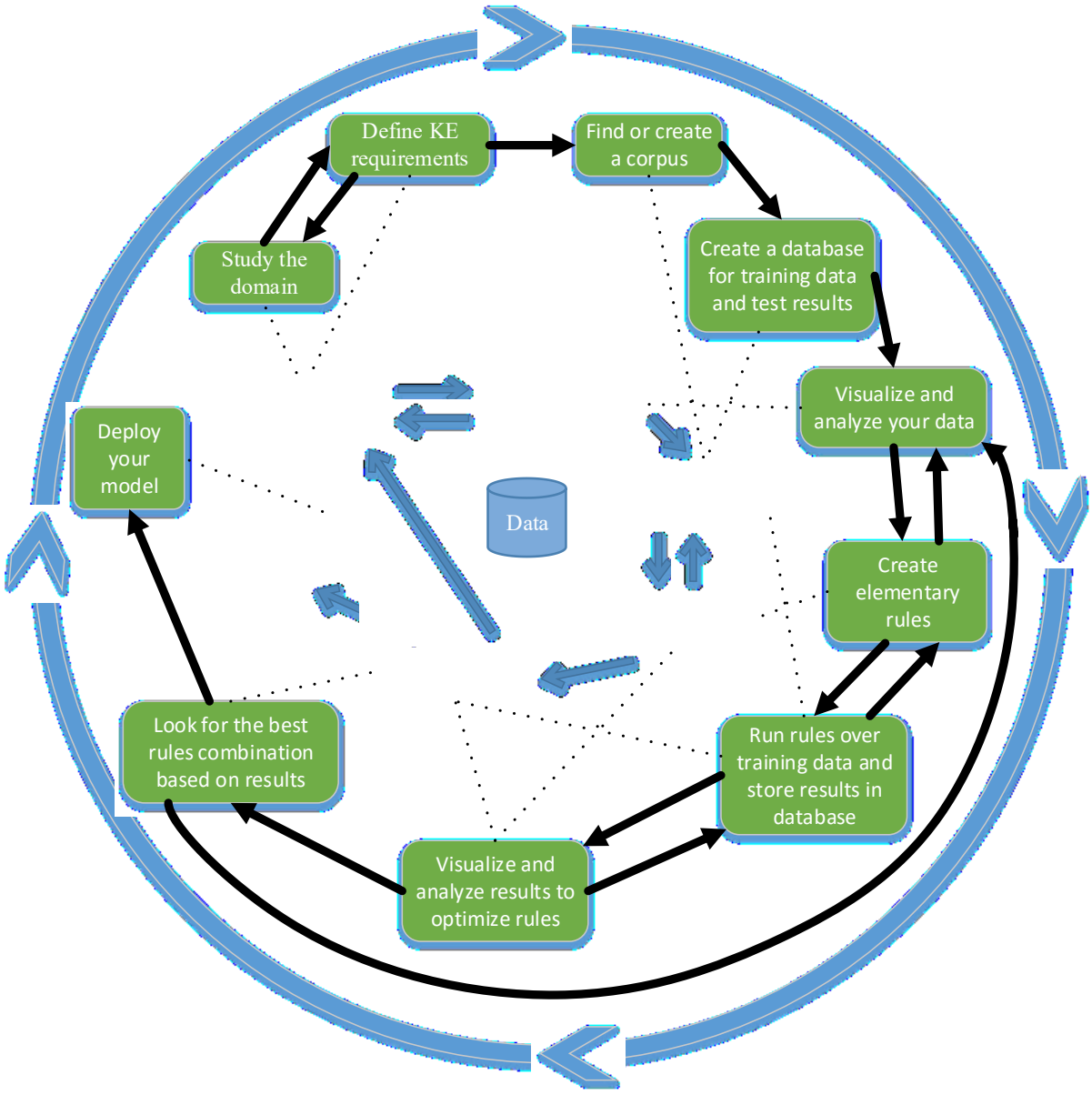


Figure 3.1: Our KEFUD process (green and black) and how it maps to CRISP-DM (in blue)

In subsequent chapters, we show how we used our KEFUD process with ARDAKE for visual KEFUD rules creation to classify sentences in the medical abstracts of the NICTA-PIBOSO corpus. Our classification results bypassed those obtained by most state-of-the-art tools and ML algorithms using the same corpus. The following subsections gives more details about each step in our process.

3.3.1 Study the domain

It is important to study the domain and have, at least, a basic understanding of the main concepts and rules before starting any KE project. This is especially true when a KE project is conducted by non-domain experts as the lack of knowledge can hinder any effective communication with domain experts who are usually the main source of information to define requirements.

3.3.2 Define/understand KE requirements

Like with any project, requirements can be adjusted and may be developed and finalized over time but starting any project without some clear requirements significantly increases the chance of failure. The requirements should at least identify the kind of knowledge to extract and the success criteria for the KE project.

3.3.3 Find or Create a Corpus (Training and Test sets)

It is possible to define KE rules and create rule-based KE models based on experience and familiarity with the domain and without a training set. This falls under the unsupervised learning methods for creating KE models. Unsupervised learning is the only option in domains where it is impossible or too costly to create training sets. However, for most domains, creating new or finding existing training and test sets is worth the efforts and costs. Having a good data set, that properly represents the domain data, allows creating and evaluating models based on supervised learning methods. Creating training and test sets can be a big challenge but many researches were done on how to simplify and improve this task due to the key role training and test sets play in supervised Data Mining and KE methods [32; 33].

3.3.4 Create a database to store the training data and your results

Databases are extremely efficient and powerful in storing and querying data. While relational databases are primarily designed to handle structured data, they have gone a long way in handling non-structured binary and textual data as well. Non-relational databases have recently become very popular due to their high capacity in storing and querying unstructured data, especially textual documents.

Storing your training set and your test results in a database gives the opportunity to use advanced database functionalities to analyse and understand the data and to analyse test results and optimize rules and models.

ARDAKE provides a generic database that can be used to store training sets and test results for KE projects that aim to identify patterns and knowledge at the sentence level as it is the case for the NICTA-PIBOSO contest.

Stored training data and test results in databases has another great benefit especially for non-technical domain experts. It enables the use of visualization tools to present the data and test results in different views and from different perspectives. In the last few years, we have witnessed the quick rise of some data visualization giants such as tableau and Qlik whose main target audiences are non-technical business users. Existing Business Intelligence leaders including SAP, Oracle, Microsoft, and IBM have also enriched their database and data analysis products with visualization.

3.3.5 Visualize and analyse your training data and test results

The English Idiom “A picture is worth a thousand words” is often used to express the fact that a complex idea can be made simple with an image. In fact, an image or a view may be worth millions of words when analysing and trying to understand a large amount of unstructured data. A good understanding of the training data and the distinguished properties of the embedded

knowledge is a crucial step before creating KE rules. This can be made much simpler using visualization tools.

Visualization tools can also be very useful in viewing and analysing the results of KE rules once they are defined and tested. Test results can be analysed without visualization but presenting test results in a visual way gives an immediate insight on where the hits and misses are and allows zooming into more details to see where the problems are and adjust the rules to fix those problems.

When analyzing data, we recommend looking for both positive and negative properties of patterns of interest. Positive properties of a pattern are properties that indicate the existence of the pattern while negative properties indicate its absence.

3.3.6 Create Elementary Rules (Typically using a visual rules editor)

Pattern identification rules are the building blocks to form more complex KE rules. Elementary KE rules can be created once the unique differentiating properties of desired patterns are identified. Inclusion and exclusion rules should be created to model the positive and negative patterns properties respectively. Rules can be expressed in so many different ways and languages but the best tool that makes rules creation available to more audience, including non-technical users, is the tool that allows the creation of a rich set of rules in a simple way. With ARDAKE, a user can create linguistic, statistical, and semantic KE rules in a simple and consistent visual way.

3.3.7 Run elementary inclusion and exclusion rules on corpus and store results in database

In order to measure the quality of rules, they should be run over the training set and have their results captured and compared with predefined training patterns. A KE rule typically has a condition and an action part. Running a rule over a training set means checking the condition

part of the rule against the training set data and applying the action part of the rule when the condition is met. An example of a KE rule is a rule where the condition is to match a sequence of a number token followed by “years old”. The action part of such rule could be to label this sequence as an age pattern.

3.3.8 Visualize and analyse test results to optimize elementary rules

As stated in 3.3.5, visualizing test results gives an immediate insight on where the problems are and help narrowing down and resolving issues in order to optimize the performance of KE rules. Analyzing test results of a KE rule is done by comparing the results of the rule with manually predefined results. Patterns identified both manually and by the rule constitute the True Positive (TP) set of the rule. Patterns identified manually and missed by the rule are the False Negative (FN) results. Patterns mistakenly identified by the rule are the False Positive (FP) results. Finally, patterns that are left out by both the rule and the manual annotation are called True Negative (TN) results. The quality of a KE rule is usually measured using its TP, FP, TN, and FN results or a formula based on them like ROC and F-Score.

3.3.9 Look for the best rules combination based on results

Some rules have a high precession with a low recall. Others have a high recall with a low precession. Rules composition has the goal of finding the rules combination that produces the right balance between precision and recall and thus optimizing the resulting F-Score. Rules for matching same pattern types can be combined by joining their conditions using Boolean operators. Combining two rules using the AND operator results in a more restrictive rule and helps eliminating FP results found in both original rules. Combining two rules using the OR operator creates a looser rule that has the TP results of both original rules. Unfortunately, combining rules using the AND operator often removes TP results from original rules and combining them using the OR operator passes the FP results of the original rules to the new rule. Therefore, combining rules should be done by carefully inspecting the commonality and disjunction between their TP, FP, TN, and FN results. Finding the best rules combination

manually is a tedious and time-consuming task even with a few elementary rules. ARDAKE includes a results-based functionality to check millions of rules combinations, in few seconds, and identify the combination that produces the best KE results.

3.4 Chapter Summary

We highlighted the most complex tasks in a KEFUD project and set our objectives to simplify these tasks by designing and developing efficient, easy to use, tools. We also described a new methodology, inspired from CRISP-DM, to guide KEFUD miners throughout the various stages of a KEFUD project while concentrating their efforts on the tasks they need to focus on at each stage.

As we show in the rest of this thesis, the methodology and tools we created greatly helped us obtain better KEFUD results than those obtained by state-of-the-art algorithms. We believe that other KEFUD projects can get the same benefits using our tools and methodology.

CHAPTER 4

N-Gram Algorithms

4.1 Chapter Overview

N-grams are widely used in NLP and text mining [34; 35; 36; 37; 38]. Frequent n-grams found in training sets can be used as keywords while looking for patterns in test sets or in actual text. N-grams should be used with care in KEFUD projects. Using the wrong set of n-grams leads to poor KE quality by increasing the number of FP and/or decreasing the number of TP results. Using long n-grams or n-grams co-occurrence helps reducing FP results but often eliminates a significant number of TP results leading to poor KE performance.

We distinguish between two types of n-grams: a positive n-gram that indicates the existing of a given pattern and a negative n-gram that indicates the absence of a pattern. We propose a new algorithm to generate positive and negative n-gram decision trees for specific patterns. We then use those n-gram decision trees to automatically generate inclusion and exclusion KE rules.

N-grams trees have been used by few researchers for language modeling. [39] used a TreeTagger to model the probability of a tagged sequence of words using a binary decision tree to estimate transition probabilities. [40] used a word-tree data structure to model textual documents where nodes represent sequences of words with their frequencies in the training corpus. Our n-grams decision trees algorithm is similar to existing DM algorithms like C5 and Classification and Regression Trees (CART) but is distinguished by its splitting criteria and stopping condition based on precision, recall, or F-Score gain. Our results demonstrate that n-gram decision trees and associated rules are more efficient than simple n-grams or n-grams co-occurrence.

4.2 What are N-Grams

An n-gram is a sequence of n consecutive tokens in a given text. Tokens can be words, letters, numbers, or even special characters. Single token n-grams (n equals 1) are known as unigrams, double tokens n-grams (n equals 2) are called bigrams, and triple tokens n-grams (n equals 3) are trigrams. The sentence “Smoking can cause cancer” has four unigrams “Smoking”, “can”, “cause”, and “cancer”. The same sentence has 3 bigrams “Smoking can”, “can cause”, and “cause cancer” and 2 trigrams “Smoking can cause” and “can cause cancer”. In general, the number of n-grams in a sentence of m words is equal to m – (n – 1).

The Oxford Web Language Model (previously Microsoft Web N-Gram Services) shows the importance of n-grams in building language models and the different applications that rely on them such as Search Query Segmentation, Word Breaking, Spell Checking, and Auto Completion. The Microsoft Web N-Gram corpus was built by parsing web pages’ contents, titles, and anchors. It includes hundreds of billions of unigrams, bigrams, trigrams, as well as n-grams of size 4 and 5.

4.3 N-Grams Limitations

In Information Extraction, an n-gram that is frequently found with other patterns in a training set can be used to determine the existence of similar patterns in test sets or in real corpora. The appearance of the same n-gram where the pattern of interest is missing can lead to a false positive result by mistakenly labeling the nearby text when the n-gram is found. The number of false positive results can be reduced by relying on the co-existence of two or more n-grams with the pattern being searched. Unfortunately, this usually increases the number of false negative results since less patterns will be identified with more coexisting n-grams. Finding the right number of coexisting n-grams to use in order to identify a given pattern is important, but not always possible, to get better IE results.

4.4 Mitigating n-grams limitations

Many researchers studied the impact of n-grams length on the precision and recall of the patterns matching results [41; 42]. Other researchers used n-grams co-occurrence and n-gram

words/tokens permutation to improve the patterns identification results. Better pattern matching can be achieved by either having a significant increase of true positive compared to the false positive results and/or by having a significant decrease of false positive results without losing much of true positive results.

Using longer n-grams or multiple n-grams co-occurrence is usually done to improve the precision by reducing the number of false positives. Unfortunately, this almost always results in eliminating a large number of true positive results leading to a lower recall and most likely a lower F-Score. A better alternative, especially in closed domains, is the use of negatively correlated or simply negative n-grams. As opposed to previously discussed n-grams that indicate the existence of certain patterns, negative n-grams indicate the absence of some patterns and can therefore be used to reduce the number of false positives. Negative n-grams can be calculated by identifying frequent n-grams that rarely or never co-exist with given patterns in a training set.

4.5 Proposed Algorithms

We propose an algorithm, for the creation for n-grams decision trees, that allows manipulating different n-grams features, including the n-gram length, co-occurrence, and the splitting criteria. Decision trees generated by our algorithm help balancing the number of true positive and false positive results to obtain the desired results based on precision, recall, or F-Score.

As shown in Figure 4.1, the ARDAKE Corpus Analyser that implements our n-gram decision trees algorithm requires a few input parameters such as the preferred n-gram length, the list of sentences containing a pattern of a given type/class in the training set, and the desired minimum correlation level between n-grams and the pattern in question. The correlation level can be calculated based on precision, recall, or the F-Score. Assigning a value that is close to 100 to the correlation level risks overfitting the resulting n-grams tree model. Increasing the values of the n-gram length and minimum frequencies parameter helps reducing the size of the resulting n-gram tree. This is important when generating trees for a large corpus as sparsity is a common issue when creating decision trees.

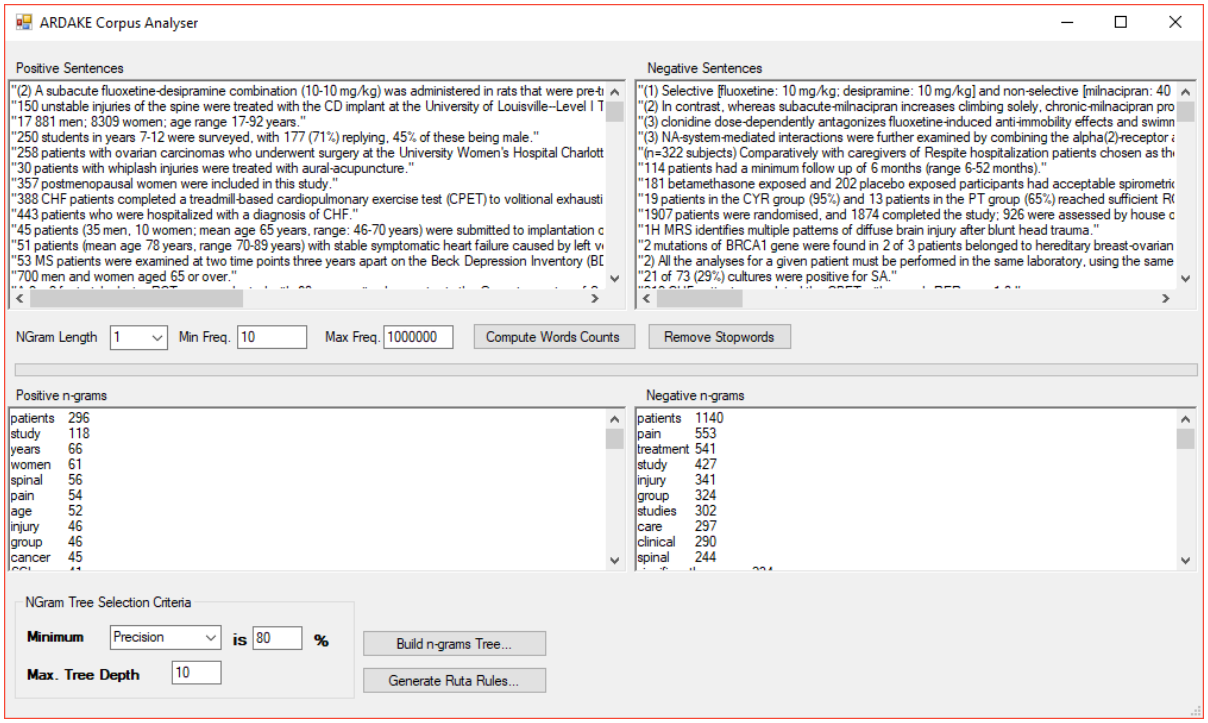


Figure 4.1: The ARDAKE Corpus Analyser

The algorithm starts by identifying frequent n-grams in both the list of positive sentences (sentences containing the patterns for the type/class of interest) and the list of negative sentences (sentences that do not contain the patterns for the type/class of interest) in the training set. This generates 3 disjoint categories of n-grams based on their level of correlation with the pattern of interest:

1- Highly positive correlated n-grams: these are n-grams that are frequently found in positive sentences but are absent or rarely found in negative sentences. N-grams in this category can be used in inclusion pattern matching rules to help increasing the number of true positive results with no or minimal increase in the number of false positives.

2- Highly negative correlated n-grams: these are n-grams that are frequently found in negative sentences but are absent or rarely found in positive sentences. N-grams in this category should be used in exclusion pattern matching rules and can play an important role in reducing the number of false positive results with a minimal or no decrease in the number of true positive results.

3- Low or no correlation n-grams: these n-grams are found in almost same proportions in both positive and negative sentences. N-grams in this category are usually ignored and considered non-deterministic when, in fact, they can have a potential value and play a key role in either increasing the number of true positives and/or decreasing the number of false positives. The next paragraph presents a real example taken from the PIBOSO training set that shows how, when combined, n-grams in this category can play an important role in improving the pattern matching results. The following section shows how our algorithm recursively combines n-grams in this category to generate additional positive and negative pattern matching rules.

To demonstrate the usefulness of n-grams in the above third category, we consider unigram "conducted" that is found in 23 positive population sentences and 53 negative population sentences of the PIBOSO training set. Similarly, unigram "patients" exists in 335 positive population sentences and 1269 negative population sentences but the two unigrams co-exists in 13 positive population sentences and only one negative population sentence. Note that the number and frequency of n-grams is normally higher in the negative population sentences since the PIBOSO training set contains 10804 negative population sentences and only 812 positive population sentences.

In the previous example, neither n-grams could be used separately to generate an inclusion or exclusion pattern matching rule. However, the combination of the two n-grams can generate a very high precision inclusion rule in case the training set correctly represents the domain corpus. Sometimes, the combination of two or more n-grams is still frequently encountered in both positive and negative sentences. To take advantage of such n-gram combinations, our algorithm incrementally adds more n-grams to the mix then checks whether an inclusion or an exclusion pattern matching rule can be generated. The end result is an XML-based n-gram decision tree in which the root node and children of no/low correlation nodes have each three child nodes:

- Positive n-grams node: Children of this node are leaf nodes for n-grams that, when combined with n-grams in their parent nodes (if any), have positive correlation with the pattern type/class of interest.
- Negative n-grams node: Children of this node are leaf nodes for n-grams that, when combined with n-grams in their parent nodes (if any), have negative correlation with the pattern type/class of interest.
- Other n-grams node: This node has a child node for each n-gram that is not included in the previous two nodes or their parents. Like the root node, each n-gram node here has three child nodes (Positive, Negative, and Other).

The next section shows how the n-grams tree is built and the order in which n-gram nodes are created and inserted into the tree. Section 4.5.2 demonstrates how inclusion and exclusion pattern matching rules can be automatically derived from the n-grams decision tree.

4.5.1 N-Grams Tree generation

The first step in building the n-grams tree is the creation of an empty root node that has three child nodes (PositiveNGrams, NegativeNGrams, and OtherNGrams). Positive and Negative sentences are then parsed to compute positive, negative, and other n-grams based on options selected by the user in the ARDAKE Corpus Analyser before generating the n-grams tree. These options include the maximum n-gram length, minimum and maximum n-gram frequency, as well as the minimum desired degree of correlation between n-grams and the pattern in question. The correlation is not calculated based on a simple count for the number of times an n-gram is found in positive and negative sentences. It is rather calculated, according to user selection, based on the precision, the recall, or the F-score value or gain. N-grams with frequencies outside the specified boundaries are dropped before the correlation calculation phase.

To classify an n-gram based on its precision, we calculate its positive and negative precisions. The positive precision for an n-gram is the ratio of the number of positive sentences containing the n-gram over the total (positive and negative) number of sentences containing the n-gram.

Likewise, the negative precision is the ratio of the number of negative sentences containing the n-gram over the total number of sentences containing the n-gram. If the positive precision is greater than or equal to the specified threshold, the n-gram is classified as a positive n-gram and a child node is created for the n-gram under the "PositiveNGrams" node. If the negative precision is greater than or equal to the specified threshold, the n-gram is classified as a negative n-gram and a child node is created for the n-gram under the "NegativeNGrams" node. In case the two previous conditions are both wrong, a child node is added for the n-gram under the OtherNGrams node. This process is recursively repeated for each node under the OtherNGrams node but the positive and negative precisions are then calculated for subsequent n-grams in conjunction with their parent n-grams. The children of each node are sorted by their level of correlation with the pattern type/class of interest.

Classifying n-grams based on recall or F-Score is done in the same manner as classifying them using the precision except that the recall or the F-Score formulas are used instead of the precision formula. Code Snippet 2 shows a partial n-gram tree for the population sentences in the PIBOSO corpus. Figure 4.2 graphically illustrates the same tree.

```
<root>
  <name_635790719492055913 Text="PositiveNGrams">
    <name_635790719492055913 Text="recruited" PCount="26" NCount="4" />
    <name_635790719492055913 Text="crossover" PCount="5" NCount="0" />
    <name_635790719492055913 Text="July" PCount="5" NCount="1" />
    …
  </name_635790719492055913>
  <name_635790719492055913 Text="NegativeNGrams">
    <name_635790719492055913 Text="significant" PCount="4" NCount="574" />
    <name_635790719492055913 Text="quality" PCount="10" NCount="221" />
    <name_635790719492055913 Text="evidence" PCount="6" NCount="202" />
    …
  </name_635790719492055913>
    <name_635790719492212171 Text="OtherNGrams">
      <name_635790719493618523 Text="patients" PCount="135" NCount="1317">
        <name_635790719493618523 Text="PositiveNGrams">
          <name_635790719493618523 Text="conducted" PCount="7" NCount="1" />
          <name_635790719493618523 Text="placebo-controlled" PCount="5" NCount="1"
/>
          …
        </name_635790719493618523>
        <name_635790719493618523 Text="NegativeNGrams">
          <name_635790719493618523 Text="symptoms" PCount="2" NCount="44" />
          <name_635790719493618523 Text="increased" PCount="2" NCount="41" />
```

```
        …
      </name_635790719493618523>
      <name_635790719493618523 Text="OtherNGrams">
        <name_635790719493618523 Text="study" PCount="61" NCount="73">
          <name_635790719493618523 Text="PositiveNGrams">
            <name_635790719493618523 Text="retrospective" PCount="6" NCount="2"
/>
            <name_635790719493618523 Text="clinical" PCount="5" NCount="2" />
            …
          </name_635790719493618523>
          <name_635790719493618523 Text="NegativeNGrams">
            <name_635790719493618523 Text="efficacy" PCount="2" NCount="7" />
            <name_635790719493618523 Text="treatment" PCount="2" NCount="6" />
            …
          </name_635790719493618523>
          <name_635790719493618523 Text="OtherNGrams">
          …
```

Code Snippet 2: n-grams decision tree sample in XML format

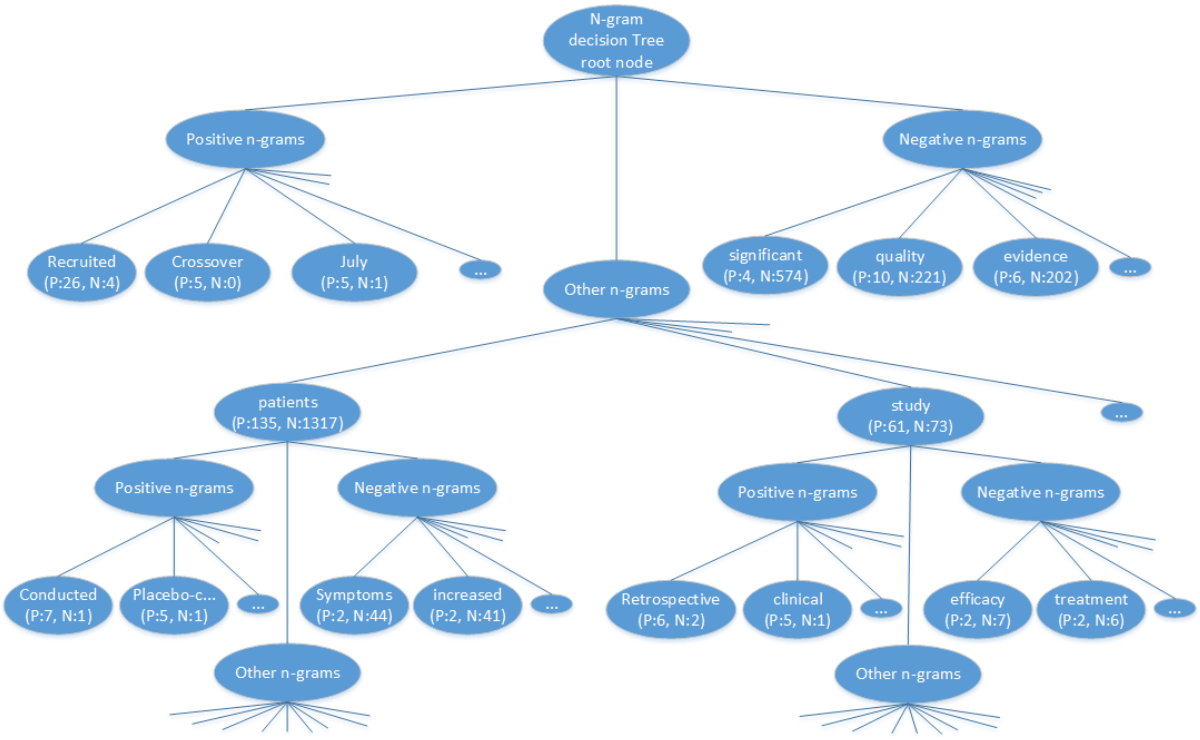


Figure 4.2: The tree representation for Code Snippet 2

4.5.2 Rules auto generation from N-Grams Trees

Generating inclusion and exclusion pattern matching rules from an n-grams decision tree is straightforward and is similar to generating rules from other decision trees. Starting from the root node, n-grams under the "PositiveNGrams" node are directly mapped into an inclusion rule. N-grams under the "NegativeNGrams" node are mapped into an exclusion rule. Finally, n-grams under the "OtherNGrams" are combined with their positive and negative n-gram descendants to create inclusion and exclusion rules respectively. The following example shows how a sample n-gram subtree is converted into inclusion and exclusion rules.

The n-grams decision tree in Code Snippet 2 generates the following inclusion and exclusion pattern matching rules:

Inclusion Rules

- A sentence that contains n-gram "recruited" or "crossover" or "July" or … is considered a positive sentence.
- A sentence that contains n-gram "patients" and ("conducted" or "placebo-controlled" or …) is considered a positive sentence.
- A sentence that contains n-grams "patients" and "study" and ("retrospective" or "clinical" or …) is considered a positive sentence.

Exclusion Rules

- A sentence that contains n-gram "significant" or "quality" or "evidence" or … is considered a negative sentence.
- A sentence that contains n-gram "patients" and ("symptoms" or "increased" or …) is considered a negative sentence.
- A sentence that contains n-grams "patients" and "study" and ("efficacy" or "treatment" or …) is considered a negative sentence.

4.6 Improving Annotation Results Using n-grams Inclusion and Exclusion Rules

Annotation results can be analysed and looked at in different ways. CHAPTER 5 demonstrates the visualization of annotation results and how it easily helps seeing what is required in order to improve these results. Improving the results is mainly done by adjusting annotation rules to

increase the number of true positive results or to decrease the number of false positive results. N-grams inclusion and exclusion rules serve the exact same purpose. The goal of inclusion rules is to increase the number of true positive results with no or insignificant increase in the number of false positive results. Similarly, the goal of exclusion rules is to decrease the number of false positive results with no or an insignificant decrease in the number of true positive results. Therefore, n-gram inclusion and exclusion rules described in this chapter play a key role in improving the annotation performance.

4.7 Chapter Summary

N-grams are powerful tools for KEFUD. They are widely used in NLP and text mining but are not used to their full benefits. When using n-grams, it is important to find the right set of n-grams for a specific pattern type. Having too many n-grams results in more FPs while having too few of them leads to less TPs. In both cases, the end KE performance suffers.

Longer n-grams or n-grams co-occurrence is sometimes used to improve the KE precision but often results in missing valid patterns. We proposed a new algorithm for creating n-gram decision trees that can be used to generate efficient KE rules for maximizing TP and minimizing FP results. Applying NLP tasks on the text, such as stopwords removal and stemming before generating the n-gram decision tree often results in a better and smaller tree. Unlike other language modeling n-gram trees that are primarily used for tagging and text proofing, our n-gram trees are specifically designed for annotation (pattern identification) and classification in textual data. Our algorithm is similar to existing DM algorithms like C5 and CART but is distinguished by its tree splitting criteria and the structure of its output n-gram decision trees.

CHAPTER 5

Prototype

5.1 Chapter Overview

In this chapter, we present the architecture of our prototype called ARDAKE and describe the tools and main functionalities we built to help KEFUD miners with their complex tasks as described in CHAPTER 3. Specialized tools can be used at different stages of a KEFUD project from Extraction, Transformation, and Loading (ETL), through the data analysis and rules/models creation and evaluation, to the deployment. Although some of our tools are designed for textual data, similar tools can be developed for other types of unstructured data such as images, audio, and video. The last section of this chapter highlights some useful UIMA Ruta extensions that we added to help KEFUD miners create richer and efficient KEFUD rules.

In the following chapters, we show how we used our prototype and tools at various stages of the methodology we described in CHAPTER 3.

5.2 Prototype Requirements

The ARDAKE prototype was defined, designed, and implemented in a way that supports our main objective which is to make rule-based KEFUD projects simpler, requiring less time and resources, while maintaining their efficiency.

In order to simplify KEFUD projects and to minimize the implementation time and number of resources needed for a KEFUD project, ARDAKE will fulfill the following requirements:

1) Non-technical domain experts should be able to create KE rules using a visual, user friendly, rule editor without having to write any code.
2) Users should have access to a set of simple integrated tools to help them with the ETL and the analysis of textual corpora.
3) Users should be able to visualize and analyse the rules execution results in order to have a better insight on how rules can be optimized.

4) ARDAKE should assist users through the automatic generation of n-gram inclusion and exclusion rules as described in 4.5.2.
5) To maintain the efficiency of rule-based KE, ARDAKE must support rules of various types including linguistic, statistical, and semantic rules. All ARDAKE rules will be created in a consistent way using a simple visual rule editor.
6) ARDAKE will assist users finding the combination of rules that produces the best results based on F-Score.
7) It should be possible to share rules with other ARDAKE users and use existing rules to define more complex ones.
8) Users should be able to run rules directly from ARDAKE or generate equivalent Ruta rules that can be used in any UIMA Ruta compatible system.

5.3 Prototype Architecture

The ARDAKE architecture is based on a set of text analysis and rule-based annotation components that complete each other to form an environment that simplifies the implementation of more accurate end to end KE solutions. ARDAKE's components can be grouped into two categories: the Corpus Manager and the Rules Manager. The Corpus Manager provides the ETL functionalities required for dealing with the corpus and includes the Corpus Extractor, the Corpus Transformer, the Corpus Loader, and the Corpus Analyser. The Rules Manager has components to deal with rules including, the Visual Rules Composer, the Ruta Generator, the Rules Results Analyser, and the Rules Combiner. Figure 5.1 shows the main ARDAKE components.

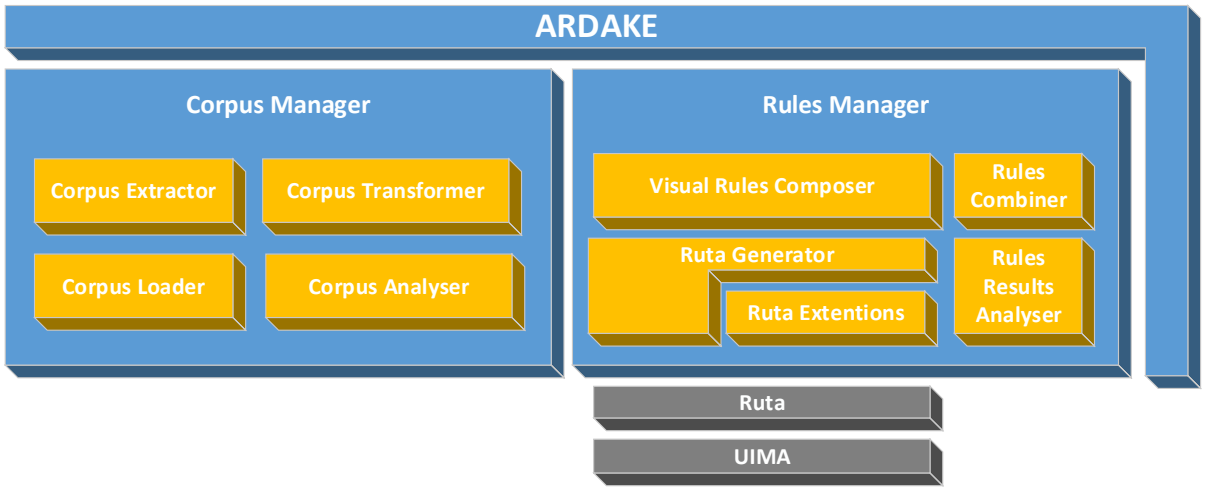


Figure 5.1: The ARDAKE architecture

We describe the main components of the ARDAKE architecture and more in the next subsections.

5.3.1 The Corpus Extractor

The Corpus Extractor helps creating local copies of corpora by downloading related files from the web or by recreating the files from a database where they have been previously imported. The Corpus Extractor is particularly helpful for making local copies of corpora whose files are available individually online at URLs that can be identified by a set of ids. This is the case of the NICTA-PIBOSO corpus where files are downloadable from the National Center for Biotechnology Information (NCBI) website using URLs in the format of "http://www.ncbi.nlm.nih.gov/pubmed/ID?dot=XML" where ID is the unique identifier of the abstract file to download in XML format. The Corpus Extractor is also useful for downloading web pages under a specific folder on a website. This could be downloading the financial news from a news website.

Reconstructing the corpus from the database is another quick way for creating a local corpus copy in case some or all local files were deleted. This can save a considerable time when a

large corpus is available on a server with a slow connection. Generating a corpus from the database is also useful in case the online corpus is no longer available.

5.3.2 The Corpus Transformer

Data transformation is an essential part of any data mining project including the mining of structured data. This is because data usually comes from different data sources and is stored in a format that the analysis of prediction algorithm cannot properly read. Since unstructured data is harder to read and analyse, this data must be passed through a transformation process that creates some sort of structure out of this data. The ARDAKE Corpus Transformer reshapes textual data in a format that makes it easier to analyse.

The Term-Document matrix is commonly used in text mining tools and algorithms [43]. It is a two-dimensional matrix where terms are on one dimension and the documents are on the other dimension. The cell values in the matrix represents the frequencies of different terms in different documents. The size of the term-document matrix can grow quickly and become huge even for medium size corpus. For large corpora, it is important to employ trimming techniques to keep the size of the term-document matrix manageable. Different techniques exist for reducing the size of a term-document matrix by eliminating stop words, low frequency terms or terms that are irrelevant for the current domain. The Inverse Document Frequency (IDF) is another technique that is commonly used to remove terms that show up in most documents with similar frequencies.

Hadoop and MapReduce are powerful tools that can be used to work with extremely large term-document matrices due to their distributed parallel storage and processing. The term-document matrix drives the logic of many text clustering and classification algorithms. These algorithms compare the frequencies of terms in the document being analysed with those in the term-document matrix in order to make their prediction, classification, clustering, etc.

The term-document matrix allows analysing the data at the document level to search, classify, or cluster documents for example. It is also possible to create association rules between terms using the term-document matrix. However, a lot of information that could be essential for many text mining projects cannot be captured or explored using a term-document matrix. This includes the paragraph and sentence level information such as the lengths and positions of sentences containing terms of interest as well as the structure of these sentences. Term-paragraph and term-sentence matrices can be used to capture and explore some information that the term-document matrix cannot capture. This obviously makes more complex the creation, storage, and analysis of all these matrices. A multi-dimensional matrix or database is a better alternative to store and analyse additional information at the paragraph, sentence, or chuck levels.

For the PIBOSO corpus, the ARDAKE Corpus Transformer extracts abstracts text from downloaded XML files that are downloaded from the NCBI website. The XML files available on the NCBI website contain the actual abstract text in a specific tag called <AbstractText>. This extraction step is required so that UIMA analysis components can parse and annotate the abstracts. The ARDAKE prototype also supports regenerating the abstracts from the database once they are imported there.

5.3.3 The Corpus Loader

Once the data is transformed into the desired format, it can be passed for analysis. To avoid repeating the same data transformation over and over, the transformed data is stored into files, databases, or data stores. This saves the transformation time and effort before every analysis. While a triple store could have been used to store transformed data, we preferred storing this data into a relational SQL database, using the ARDAKE Corpus Loader (Figure 5.2), for a number of reasons. The main reason behind using a relational SQL database is because most visualization tools are designed for tabular data and relational database. In addition, it is easier to manually or dynamically create SQL statements, functions, and stored procedures than creating SPARQL queries.

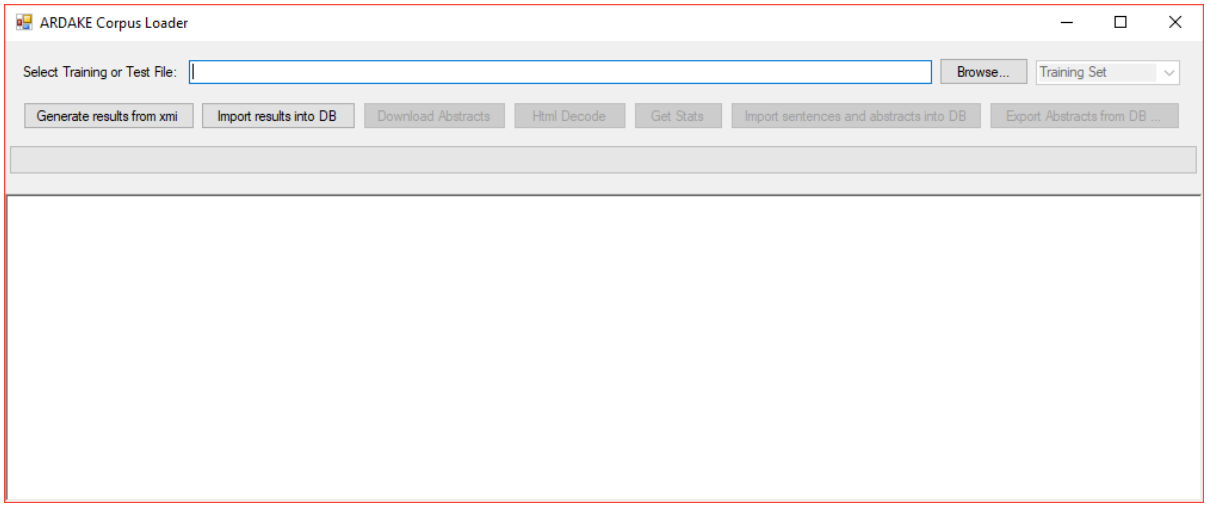


Figure 5.2: The ARDAKE Corpus Loader

5.3.4 The Corpus Analyser

The Corpus Analyser uses the functionalities built into the ARDAKE to identify common properties such as frequent lengths and position ranges for sentences of specific types. A key component in the ARDAKE Corpus Analyser is the n-gram decision tree generator (see CHAPTER 4) that produces inclusion and exclusion n-gram trees in a well-defined XML format. The ARDAKE Corpus Analyser allows users to automatically generate inclusion and exclusion rules, based on n-gram trees, using the "Generate Ruta Rules" button (Figure 4.1). N-gram decision trees and other common properties found using the Corpus Analyser constitute the basic elements for building the initial set of annotation rules in ARDAKE.

5.3.5 The Rules Composer

ARDAKE rules have the same general structure as the UIMA Ruta rules *PATTERN+ {CONDITIONS? -> ACTION+}* where each rule has one or more patterns, an optional set of conditions also called filters, and one or more actions. The Patterns part of a rule determines the initial set of tokens to match. Actions are only applied on tokens identified in the first step

and where all conditions (if any) are satisfied (i.e. tokens that pass all filters in the Filters node). Since conditions are optional, actions are applied to all tokens matching the patterns part when no condition is specified. Having a rule without a pattern or an action part is useless because a rule without a pattern is not applied to any token and a rule without an action would select and filter patterns without making any changes. If a rule has multiple patterns, then only tokens matching the sequence of those patterns in the same order are considered for the rule. Figure 5.3 shows two examples of pattern sequences (one for Age and one for AgeRange tokens).

The Rules Composer (Figure 5.3) is a visual rules editor that greatly simplifies the creation of annotation rules. It makes it possible for even non-technical users, including domain experts, to create and maintain advanced and powerful annotation rules.

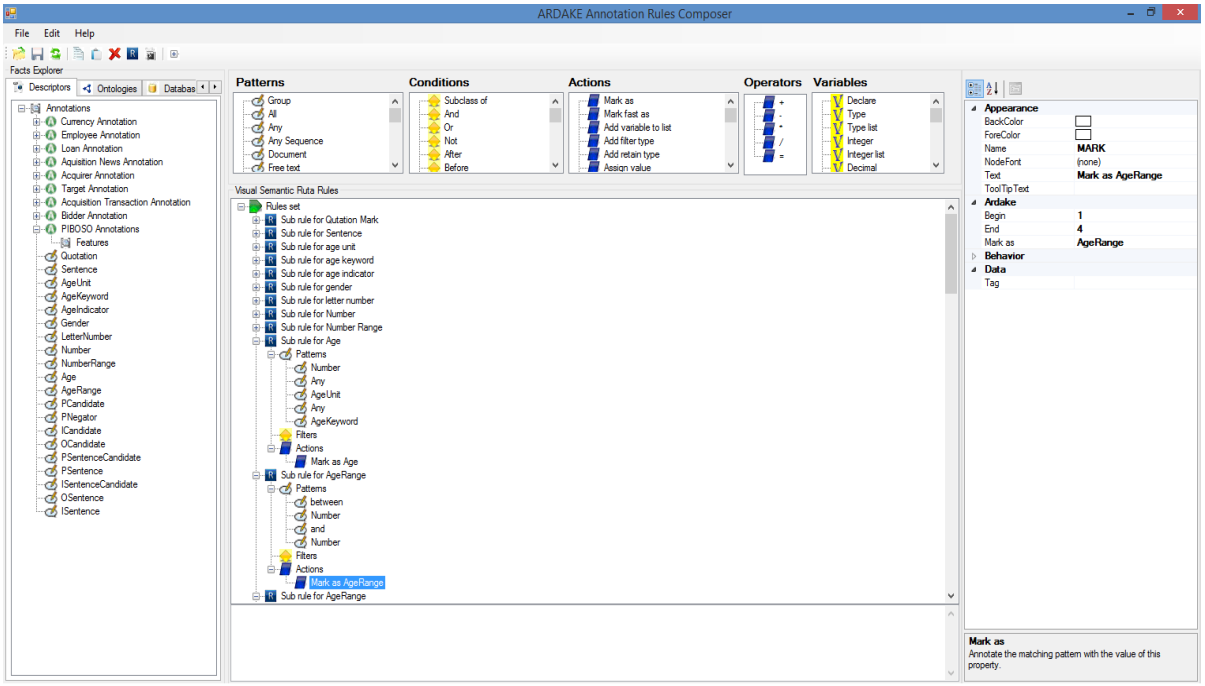


Figure 5.3: The ARDAKE Annotation Rules Composer

When a new annotation library (rules set) is created, an empty rule is added with two mandatory nodes (Patterns and Actions) and an optional node (Filters). Users define an annotation rule by dragging items from the list boxes presented at the top of the ARDAKE

Annotation Rules Composer and dropping them into corresponding nodes of the rule. Items in the Patterns list box are used to populate the Patterns node of a rule. Similarly, items in the Conditions list box are used to populate the Filters node and items in the Actions list box are used for the Actions node of a rule.

If an action for creating a new annotation type is added to the Actions node of a rule, the resulting annotation type gets automatically added to the Annotators list to the right of the Rules Composer. These annotation types can then be used as patterns when creating new rules.

More complex patterns can be defined by grouping items using the “Group pattern” that allows combining other patterns with logical operators. We discuss the rules structure and the different rules elements in more details in Section 5.5.

Users can create new rules by clicking the “R” button from the toolbar. The “X” button in the toolbar allows deleting a rule or a specific element inside a rule. The script icon beside the “R” button in the toolbar generates the Ruta script for all the rules in the current rules set.

ARDAKE supports many rule types including textual, linguistic, statistical, and semantic rules. ARDAKE rules are presented in a tree structure where each pattern, condition, and action is a node in the tree. A property grid allows setting the properties of any selected node as shown in Figure 5.4.

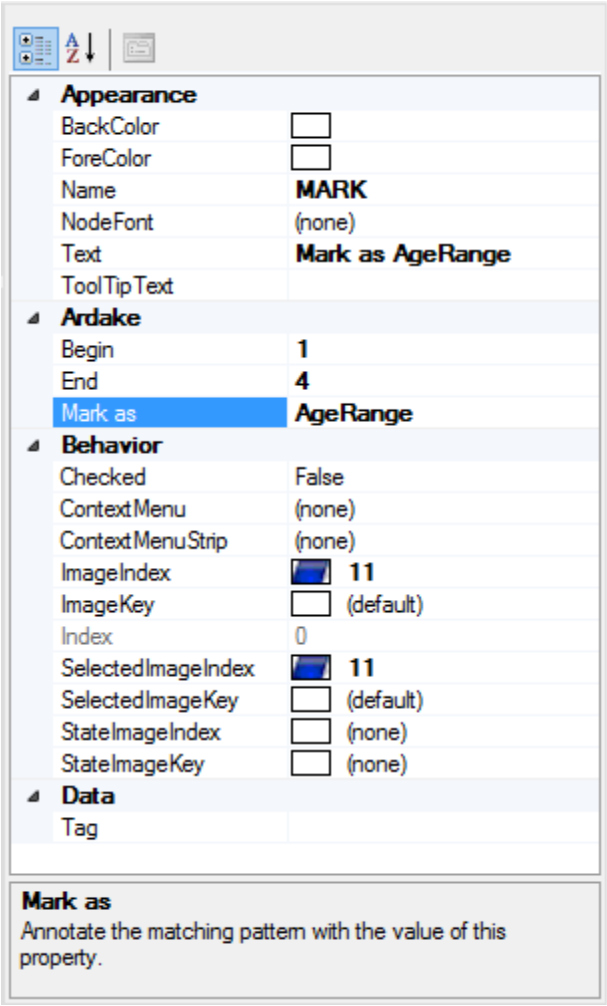


Figure 5.4: ARDAKE properties grid for an annotation action

5.3.6 The Ruta Generator

The Ruta generator converts ARDAKE rules into UIMA Ruta scripts so that they can be executed in any UIMA enabled environment. What makes this conversion simple is that ARDAKE visual rules follow the same structure as Ruta rules where each rule has one or more patterns, an optional set of conditions, and one or more actions. The Ruta generator works in a polymorphic way to convert ARDAKE visual rules into a Ruta script. It does this by navigating the rules tree structure from the root down while prompting each node for its equivalent Ruta

code. Table 5.1, Table 5.2, and Table 5.5 at the end of this chapter show Ruta scripts generated by the ARDAKE Ruta Generator for different kinds of KEFUD rules.

5.3.7 The Rules Results Analyser

The Rules Results Analyser helps rules designers improve the quality of their rules by showing the different kinds of problems in the results produced by those rules. The Rules Results Analyser compares the rules results (results produced by running the rule over the corpus test set) with the set of predefined results (manually created results) in order to create four sets of matches (True Positives, False Positives, True Negatives, and False Negatives). Rules designers can use this information to adjust the patterns, conditions, and/or actions of their rules in order to improve their accuracy. For example, a rules designer can decide to tighten the conditions of a rule if it yields too many false positives.

A rule's performance is determined by the results it produces when executed over a test set. To measure the F-Score of a rule, its TP, FP, and FN should first be calculated. These measurements along with the TN can be analysed to figure out where and why a rule is poorly performing. For example, a rule that generates lots of FP could give an indication that the rule's condition is too loose and needs to be more restrictive. Similarly, a rule that produces lots of FN may require to match on more patterns and/or have its condition(s) relaxed to add more relevant results.

Analysing the results of different rules also helps determining what logical operators to use for combining specific rules in order to obtain compound rules with higher F-Scores. Rules that share most TP but only few FP results should be combined using the logical AND operator while those that have more FP and less TP in common are better combined using the logical OR operator.

Visualizing “rules results” gives an immediate insight into their quality by graphically showing the proportion of each measurement. This could be useful to determine which rules to combine using what logical operator in order to obtain more accurate results.

Figure 5.5 and Figure 5.6 show the visual representations of the results of a (high precision, low-recall) rule and a (low precision, high recall) rule respectively. The portion between the solid lines (relevant portion) in the pie chart represents all relevant results for a specific query. The portion between the dashed lines (rule results portion) represents the results returned by a rule. The intersection between the relevant portion and the rule results portion is the TP portion of the rule. FP is the sub portion of the rule results portion that is not part of the relevant portion while FN is the sub portion of the relevant portion that is not part of rule results portion. Finally, the TN portion is the portion that is outside the relevant and the rule results portions.

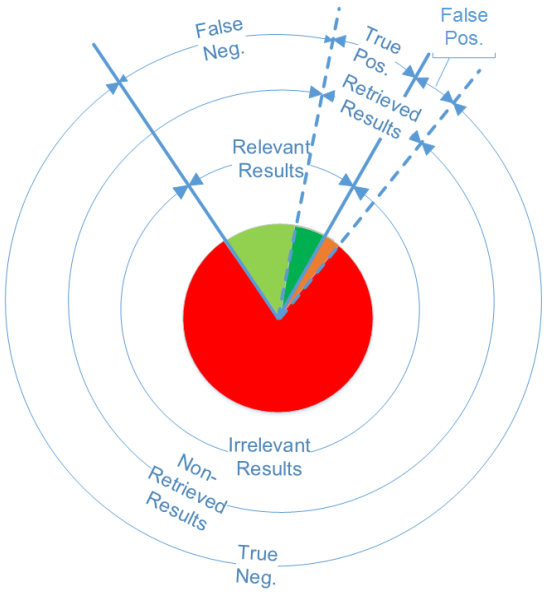


Figure 5.5: R. 1- High Precision-Low Recall

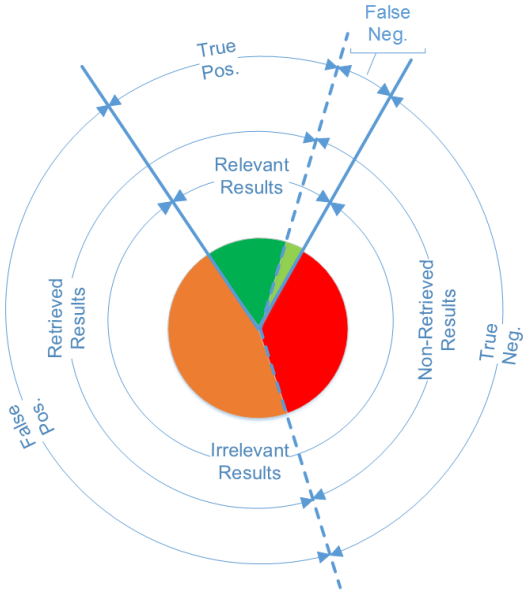


Figure 5.6: R. 2- Low Precision-High Recall

Combining the above rules using the logic AND operator is likely to produce a rule with a higher precision and a lower recall than both rules as shown in Figure 5.7. Combining the same rules using the logic OR operator is likely to produce a rule with a lower precision and a higher recall than both rules as shown in Figure 5.8.

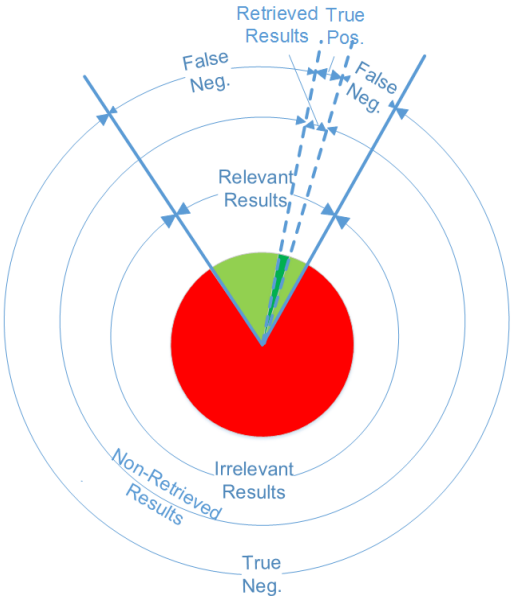


Figure 5.7: Results of Rule1 AND Rule2

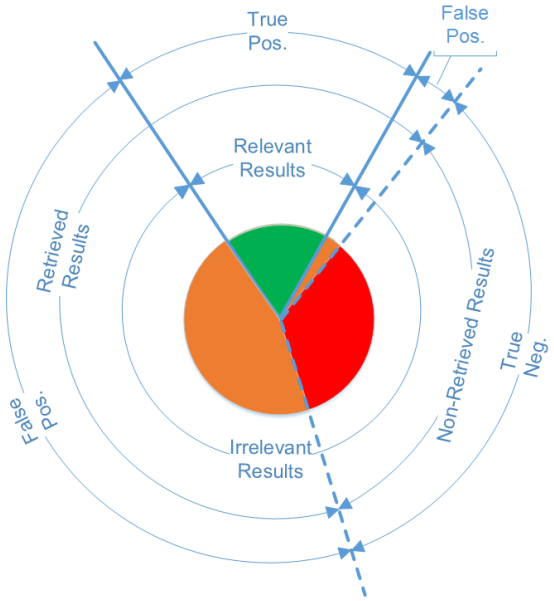


Figure 5.8: Results of Rule1 OR Rule2

Manual rules composition can be done on very small rule sets. Automatic rules composition should be considered when dealing with more than just a few rules.

5.3.8 The Rules Combiner

The Rules Combiner starts with an initial set of rules for extracting a specific pattern. Each rule is executed separately over the training set and is saved along with its matching results into a relational database where all training and test data is stored and relevant results are predefined. Having all this information in a relational database greatly simplifies the analysis of rules results using SQL and data mining tools. It also makes it trivial, using SQL queries, to calculate the TP, TN, FP, and FN of any rule and to compare and contrast the results of different rules.

The biggest benefit of storing rules results is in enabling the automatic generation of the results and the calculation of the F-Score for the combination of any subset from the initial rules without having to run the rules combination over the training set. This allows the evaluation

of millions of rules combinations in few seconds instead of spending hours to only run few rules combinations over the training or the test set. Fig. 7 shows how the results and F-Score of a composite rule (R1 OR R2) can be calculated using the results of its constituent rules. The left part of the array represents the relevant results while the right part represents the irrelevant ones. A 1 in the left part indicates a TP and a 0 in the left part indicates a FN. Similarly, a 1 in the right part indicates a FP and a 0 is for a TN. The example in Figure 5.9 also shows how combining two rules can result in a better F-Score.

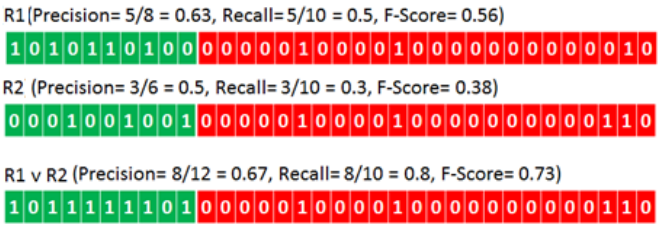


Figure 5.9: Calculating the results of R1 OR R2

Our automatic rules combining tool (Figure 5.10) combines each pair of rules in the initial rules set using the AND and the OR logical operators. Composite rules with F-Scores higher than their constituents are added to the initial set of rules and then combined with other primitive and composite rules to get more complex rules with even higher F-Scores. The tool allows users to select the level of granularity for adding composite rules. For example, a user can specify that only rules combinations with an F-Score that is at least 3% higher than their constituent rules are accepted. The process stops when no more rules are added to the rules set or when a specified number of combinations has been generated and tested. The tool returns the rules combination with the highest F-Score.

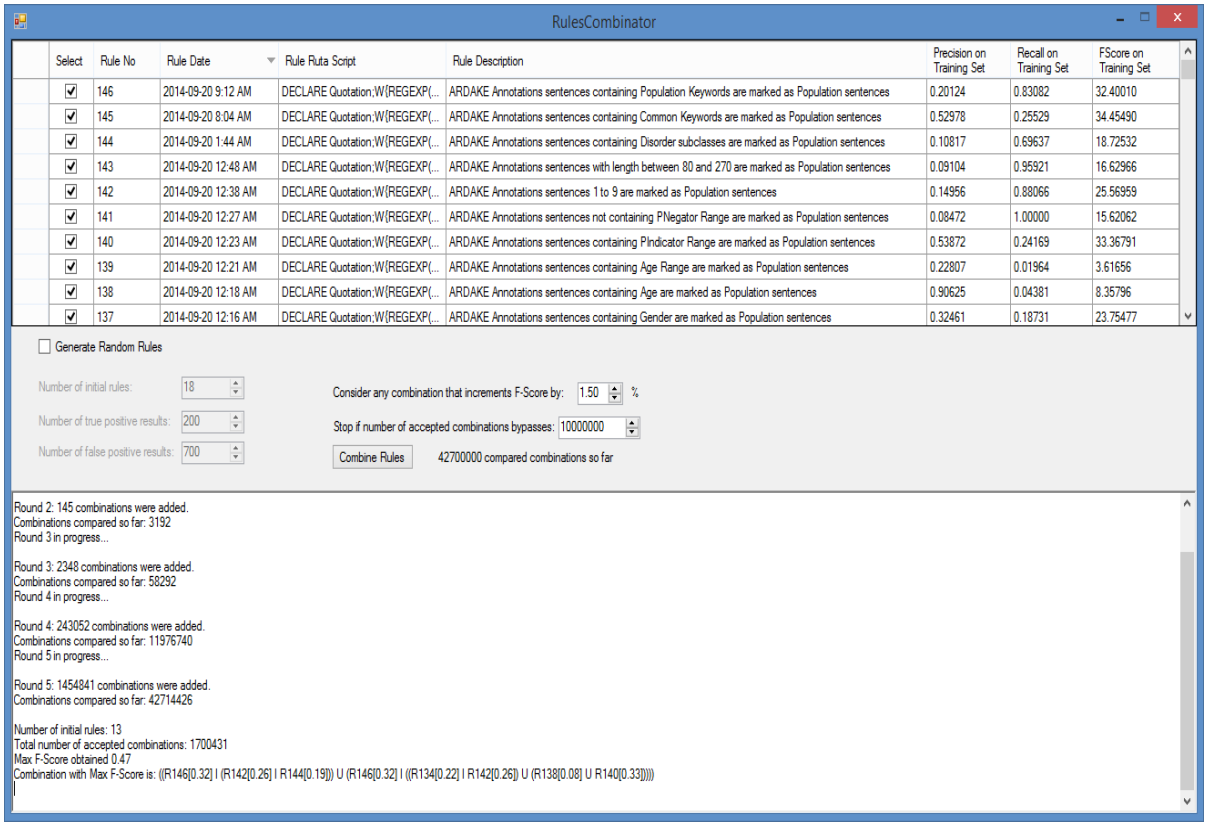


Figure 5.10: Results-Based rules combiner

We simulated thousands of rules combinations scenarios using various number of initial rules at different granularity levels, and different training sets sizes. With the exception of some rare extreme cases, our rules combiner produced rule combinations with F-Scores that are significantly higher than those of the initial rules [44].

5.4 The ARDAKE Database

To help with the analysis and to get a better understanding of the training data, ARDAKE provides a database where corpus data can be imported and analysed. The ARDAKE database schema (Figure 5.11) is simple but offers many useful analysis and comparison functionalities.

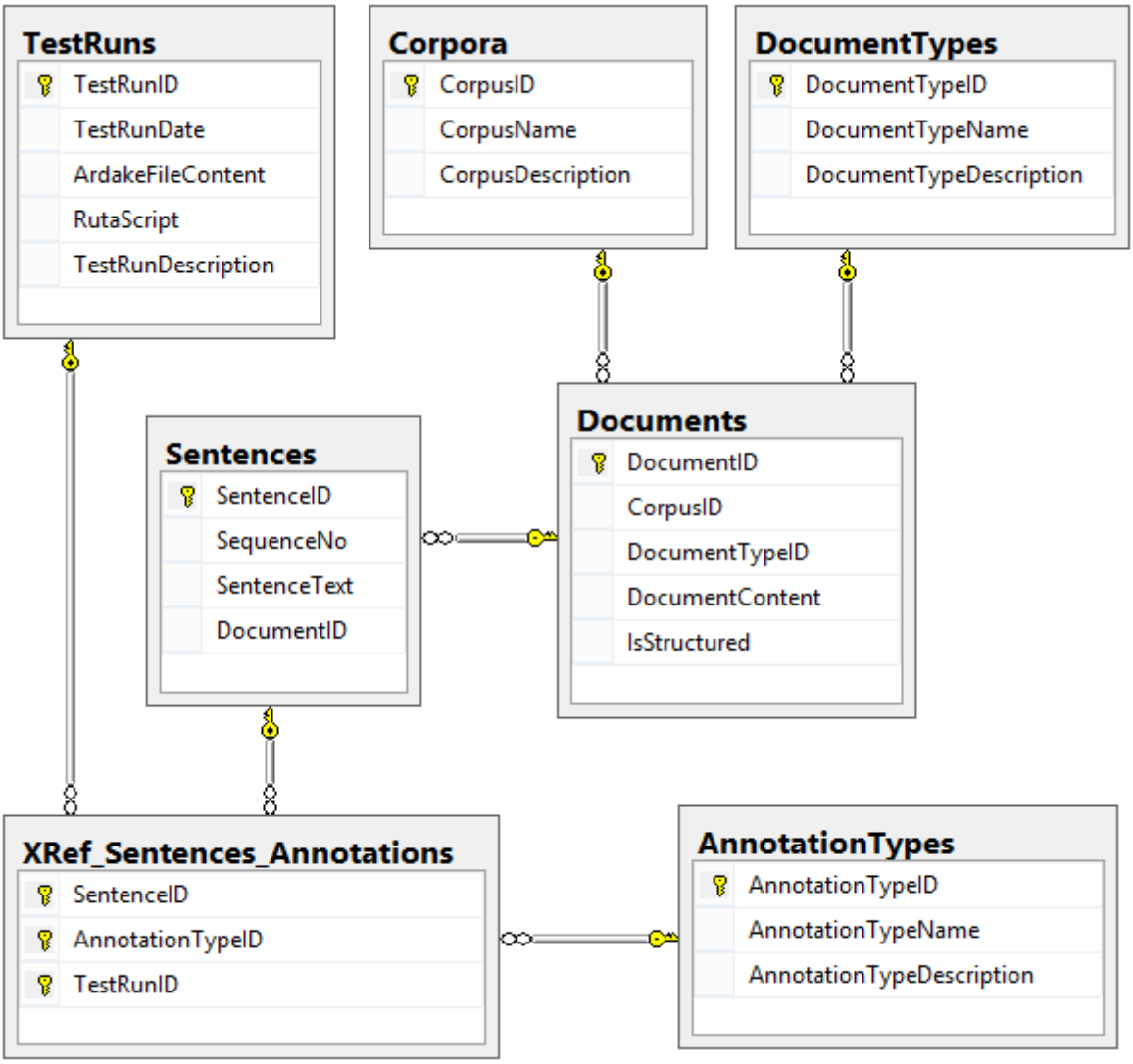


Figure 5.11: The ARDAKE database schema

The Corpora table contains names and descriptions of corpora imported into the ARDAKE database. A corpus is identified by its corpus id and is assigned a user-friendly name and description. Documents are stored in the Documents table and are assigned to Corpora through the CorpusID foreign key. Each document has a type that is determined by the DocumentTypeID foreign key.

The DocumentTypes table defines the different document types in corpora such as “Training”, “Validation”, and “Test”. The DocumentContent column of the Documents table is a Binary

Large Object (BLOB) to store the content of documents. Only text documents are currently supported by ARDAKE but other formats can be easily added. Storing documents content is not mandatory but is useful to avoid downloading the corpus over and over from the internet or from other storage locations.

Documents content can still be reconstructed from the Sentences table where all sentences are stored and linked back to their original documents. However, reconstructing a document using its sentences does not guarantee having an exact match with the original documents as line breaks/empty line might be lost. This can be adjusted by extending the database schema to store information about paragraphs.

The SequenceNo field of the Sentences table contains the location of each sentence within its parent document. This information is helpful as some annotation types are commonly found in the first few sentences while others usually reside in the last few sentences of a given document.

The schema in Figure 5.11 was specifically designed to help analysing and measuring the performance of annotations done at the sentence level such as the case for the PIBOSO corpus. For more granular annotations (i.e. annotations done at the words or other tokens levels), a further breakdown must be done at the database schema level to include those annotations. The TestRuns table is where information about each annotation test exercise is stored. The XRef_Sentences_Annotations table is where annotations created by different test runs are stored. It links the TestRuns table to the Sentences table making it possible to easily get, at any time, the list of all annotations for any test run as well as the annotation type(s) that was assigned to any given sentence during a test run.

The ARDAKE database allows comparing different annotation algorithms and tools by comparing annotations produced by those algorithms and tools to see where they match and where they do not. Evaluating the quality of annotation algorithms and tools become a simple task once the manual annotations are added as a special test run. Comparing and contrasting

annotations produced by a test run with the manual annotations gives the number of True Positives, False Positive, True Negatives and False Negatives for that test run, which can be used to calculate the F-Score or other evaluation measurements for the test run.

The ARDAKE database provides SQL functions and stored procedures for comparing different test runs including comparing test runs with manual test runs and calculating their performance measurements such as their precision, recall, and F-Scores.

5.5 Main functionalities

5.5.1 Rules structures

All ARDAKE rules have the same structure that includes three parts (patterns, conditions, and actions). The patterns section of an ARDAKE rule can contain any combination of patterns to match when the rule is applied. A pattern can be the whole document being analysed, a paragraph, a sentence, a word, a token, or any combination of these. Annotation types created by existing rules can also be used as patterns when defining new rules. This is a powerful feature since it allows the creation of compound and more interesting annotation rules based on the results of previously defined rules. For example, assuming we have three different annotation rules for matching dates, times, and locations respectively. A new rule can then be defined to create meeting annotations when a sequence of date, time, and location annotations is detected in the same sentence. ARDAKE patterns can be grouped together using the AND and OR logical operators.

Conditions in ARDAKE rules are used to filter the patterns on which the rule actions are applied. If a rule has no conditions, the rule actions are applied on all tokens matched based on patterns defined in the patterns section of the rule. An example of a condition could be to consider meeting patterns for which the date is between two given dates. An ARDAKE rule can have as many conditions as needed.

ARDAKE users can specify one or more actions to apply on annotations matching the patterns and satisfying the conditions of a rule. ARDAKE supports a large number of annotation actions that we describe in section 5.5.2.3. The “Mark” and the “Mark Fast” actions for creating new annotations are probably the most important ones. Another interesting action is “Mark Score” that allows assigning a score to a matched pattern for its likelihood to be an annotation of a certain type. Other rules can use the score action to increase or decrease scores based on specific conditions. This helps rules designers to defer the creation of annotations until multiple rules are evaluated and consider patterns whose scores are above a specific threshold.

5.5.2 Rules elements

ARDAKE rules elements are grouped in three main categories (Patterns, Conditions, and Actions). ARDAKE has two additional categories (Operators and Variables) for advanced users to create more complex rules.

5.5.2.1 Patterns

ARDAKE has many built-in elementary patterns to match basic tokens in text such as words, spaces, exclamation marks, and other special characters. This includes “Space”, “Line Break” for matching a single space and a line break character respectively. The “White Space” pattern is the parent of both “Space” and “Line Break” and can therefore be used to match either a single space or a new line character. The “Exclamation mark”, “Period”, and “Question mark” patterns can be used to match respective characters in text. The “Sentence End” matches any of the previous three patterns. Similarly, the “Comma”, “Colon” and “Semi Colon” help matching respective punctuation marks. To match any punctuation mark including sentence end tokens, the “Punctuation Mark” pattern can be used. Other granular patterns include “Symbol”, “Non-breaking space”, and “Ampersand” for matching any special character, an html non-breaking space, and html ampersand “&” respectively. “HTML and XML elements” helps matching any html or xml tag. ARDAKE also includes coarse patterns such as “Lower case word”, “Begins with capital”, “Capital word” that are sub-patterns of the “Word” pattern. The “Number” pattern matches any numeric token in a text. The “Any”

matches any of the patterns described in this section except html and xml tags. “All” matches any of the patterns described in this section including html and xml tags. ARDAKE provides even more complex patterns like “Free text” that allows rules designers to specify free text to match in a document and “Any Sequence” to match a sequence of any number of consecutive tokens for a given pattern. Finally, the “Document” pattern matches the whole text of a document and is usually used with the Markfast action described in 5.5.2.3.

5.5.2.2 Conditions

In most cases, many tokens matched by the patterns part of a rule are irrelevant. Rules can have any number of conditions to filter irrelevant tokens and only keep those on which actions should be taken. Multiple conditions can be combined using the “And” and “Or” logical operators. Negation is also possible using the “Not” operator. Tokens can be excluded based on their position within their document, their sentence or their relative position compared to other tokens. This can be done using various position-based conditions including “Before”, “After”, “Between”, “Last”, “Near”, and “Is at position”. An example of this would be to only consider time tokens that show up after a date token in a sentence. Tokens can also be filtered based on whether or not they contain, start, or end with other tokens. This is enabled by the “Contains”, “Starts with”, and “Ends with” conditions respectively. Similarly, a token can be checked to see if it is part of another token using the “Part of” and “Part of but not equal to” conditions. It is possible to discard less frequent tokens using the “Has more annotations” condition that compares tokens for two annotation types and evaluates to true for more frequent tokens within a specified window (sentence, paragraph, or a document).

Annotations can be filtered based on their types using the “Is of type” condition. This is useful when some tokens are marked with multiple annotation types simultaneously but actions should be taken on only those with a specific annotation type.

The “Has feature with value” condition allows filtering annotations based on their features (properties). Given annotations of type person with an age feature, the “Has feature with value” can be used to eliminate all persons where the age is less than 50 for example.

Annotations can be filtered based on their frequencies within a given window. “Total count”, “Count”, “Context count”, and “Current count” can be used to eliminate annotations based on their number of occurrences within the document, or other specified text windows.

The “Enough conditions” is used when multiple conditions are specified and actions should be taken on those fulfilling a certain number of conditions but not necessarily all of them. The user should specify the minimum number of conditions that must evaluate to true in order to apply the actions.

Advanced users can do more complex filtering using the “Matches regular expression” conditions where the user specifies a regular expression to keep only annotations matching this regular expression.

The “Score” condition is one of the most interesting conditions as it allows to check the value assigned to each token, using “Mark Score” action discussed later, for the likelihood of being of a certain type. This allows the creation of fuzzy rules where the score of a given token determines its membership degree to different pattern types. The score condition also allows deferring the annotation creation action until a certain number of rules have been evaluated and only create annotations for tokens whose score is higher than a given threshold. See section 5.5.3 for examples about this condition and other conditions discussed here.

5.5.2.3 Actions

ARDAKE has a long list of actions that can be applied to tokens that match the patterns part and satisfy the conditions of annotation rules. Various actions, starting with the Mark keyword, exist for creating or identifying new annotations in different ways. The “Mark as” creates annotations for tokens matched by the patterns and satisfying the conditions of a rule. “Mark fast as” create annotations for tokens matching items listed in an external file. When the “Mark fast as” action is used, users should use the custom properties grid as shown in Figure 5.12. In

this example, MonthList.txt is a text file containing the list of all months where each month is entered on a separate line.

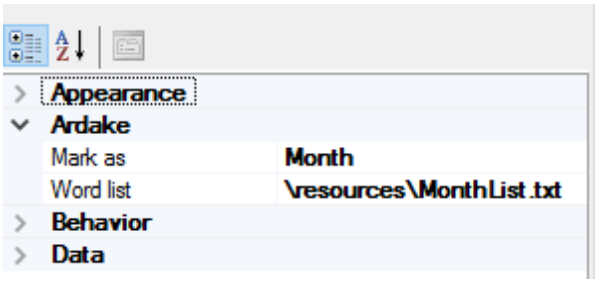


Figure 5.12: Properties for the "Mark fast as" action

“Mark score” is another interesting action that assigns a score to a token rather than creating an annotation for it. Different rules can increase or decrease the score of tokens based on various conditions. Later on, tokens with specific scores can be considered for further actions.

When multiple tokens of the same type exist within a given text window, an annotation can be created for the first or the last token using the “Mark first” or the “Mark last” actions respectively. The “Mark once” action creates a new annotation for a matching token only if this same token does not already have existing annotations.

The “Set features” action can be used to create or update annotation features (properties). To create an annotation and set its features at the same time, use the “Create annotation with features” action.

Many transient annotations are usually created during the annotation process. These are usually used in subsequent rules to create more complex annotations but are not needed once those rules are executed. For example, in order to create annotations for meeting patterns, annotations of type date, time, and location are first created then combined to identify meeting tokens. Once the meeting annotations are created, annotations of type date, time, and location may no longer be needed and can therefore be deleted to reduce the total number of annotations. Deleting annotations can be done using the “Remove annotation” action.

Duplicate annotations created by different rules for the same tokens can be cleaned using the "Remove duplicates" action.

5.5.3 Rules examples

In this section, we demonstrate the creation of a number of annotation rules in ARDAKE and present their equivalent Ruta script to show the simplicity of ARDAKE compared to the Ruta language.

5.5.3.1 Basic Rules

Figure 5.13 shows how annotation rules can be visually created for identifying age and age range tokens in text and creating annotations for those tokens. Every rule is built by dragging elements from the patterns, conditions, and actions list at the top of the ARDAKE's Rules Composer and dropping these elements into their respective nodes of the rule. User defined annotation types are automatically added to the list of annotators on the left side of the Rules Composer and can be used as patterns while defining subsequent rules.

All rules in Figure 5.13 (A) except Number use the "Mark fast" action to create annotations of type LetterNumber, AgeKeyword, and AgeUnit respectively. These rules reference external text files where possible values are listed. An age keyword is either "old" or "of age". Possible values for an age unit include "year", "years", "yr", "yrs", "month", "months", "week", "weeks", "day", and "days". The Number rule uses the "Group" pattern to define a Number pattern as either a digital or a letter number. The "Group" pattern allows combining other patterns using logical "And" and "Or" operators.

Figure 5.13 (B) shows how annotation types defined in previous rules can be used as patterns while defining new rules. The Age rule define an age pattern as a sequence of a number followed by any token followed by an age unit then an optional token and an optional age keyword. To define a pattern as optional in ARDAKE, set its Mandatory property to false as shown in the Properties grid below the Age rule in Figure 5.13 (B).

The Age rule in Figure 5.13 (B) create age annotations for various tokens including “15 years old”, “twenty five years of age”, “2-yrs-old”, and more.

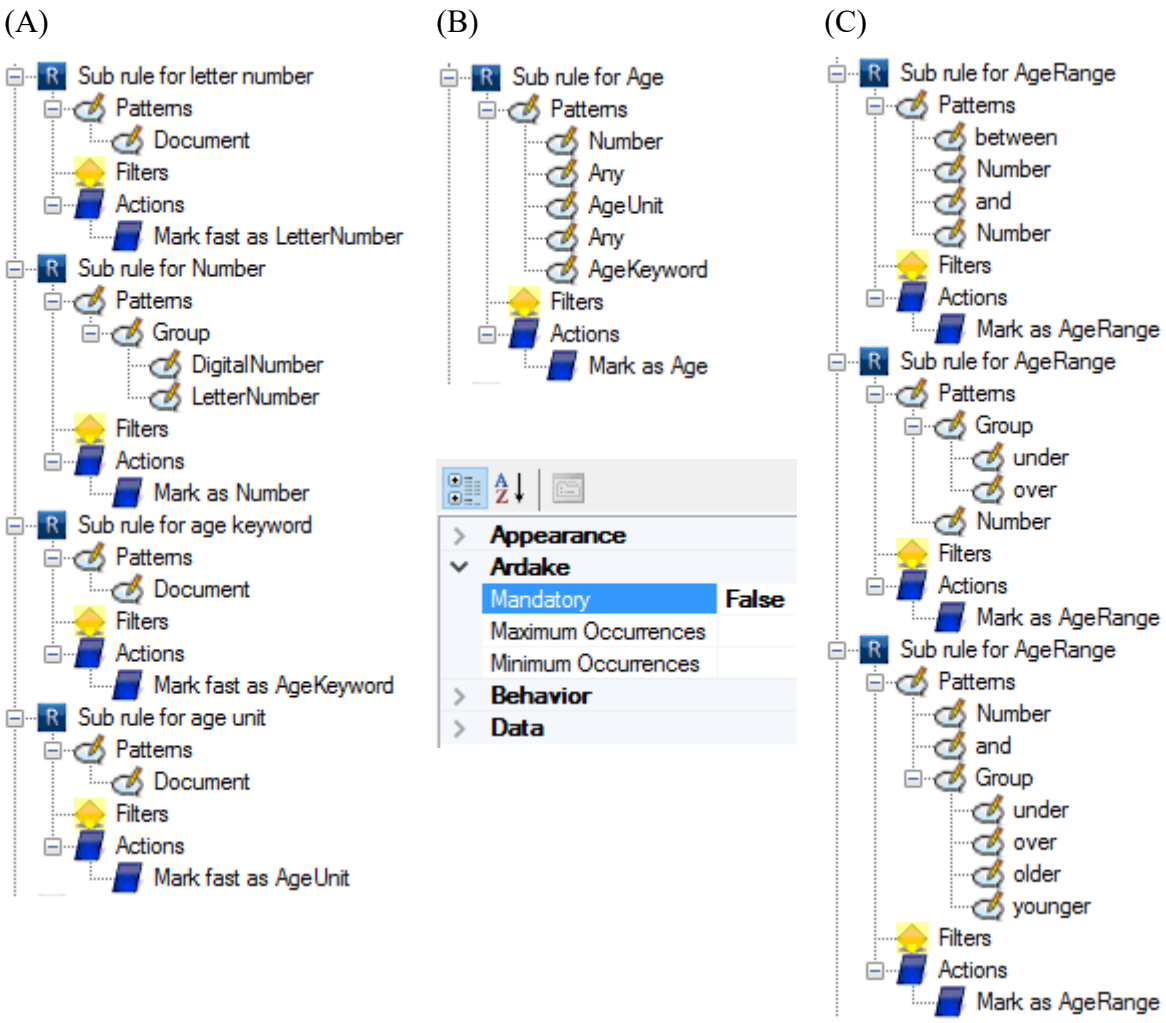


Figure 5.13: Age and Age Range rules in ARDAKE

Figure 5.13 (C) shows three different rules to define age range patterns in various shapes. This allows identifying various age pattern tokens in text including “between 10 and 12”, “between fifty and fifty five”, “under 2”, “over four”, “16 and under”, “eighteen and over”, and more.

Table 5.1 below shows equivalent Ruta code, generated by ARDAKE, for rules in Figure 5.13.

Table 5.1: Equivalent Ruta for rules in Figure 5.13

(A)	DECLARE LetterNumber; WORDLIST LetterNumberList = 'LetterNumberList.txt'; Document{->MARKFAST(LetterNumber, LetterNumberList)}; DECLARE Number; (NUM | LetterNumber){->MARK(Number)}; DECLARE NumberRange; Number ANY Number{->MARK(NumberRange, 1, 3)}; DECLARE AgeKeyword; WORDLIST AgeKeyword = 'AgeKeyword.txt'; Document{->MARKFAST(AgeKeyword, AgeKeyword)}; DECLARE AgeUnit; WORDLIST AgeUnit = 'AgeUnit.txt'; Document{->MARKFAST(AgeUnit, AgeUnit)};
(B)	DECLARE Age; Number ANY? AgeUnit ANY? AgeKeyword?{->MARK(Age, 1, 5)};
(C)	DECLARE AgeRange; "between" Number "and" Number{->MARK(AgeRange, 1, 4)}; ("under" | "over") Number{->MARK(AgeRange, 1, 2)}; Number "and" ("under" | "over" | "older" | "younger"){->MARK(AgeRange, 1, 3)};

5.5.3.2 Advanced Rules

Creating advanced rules in ARDAKE is as easy as creating simple ones. Figure 5.14 shows an example for creating a PIBOSO_Disease semantic annotation rule and three other rules for assigning scores to sentences satisfying certain conditions for their likelihood to be PIBOSO population sentences. The “PIBOSO Diseases” rule in Figure 5.14 (A) uses the semantic “Subclass” condition to create annotations for any combination of 1 to 5 words that matches

any concept that directly or indirectly inherits from the “Spinal Cord Diseases”, “Brain Injuries”, or “Demyelinating Diseases” concepts in the MESH ontology.

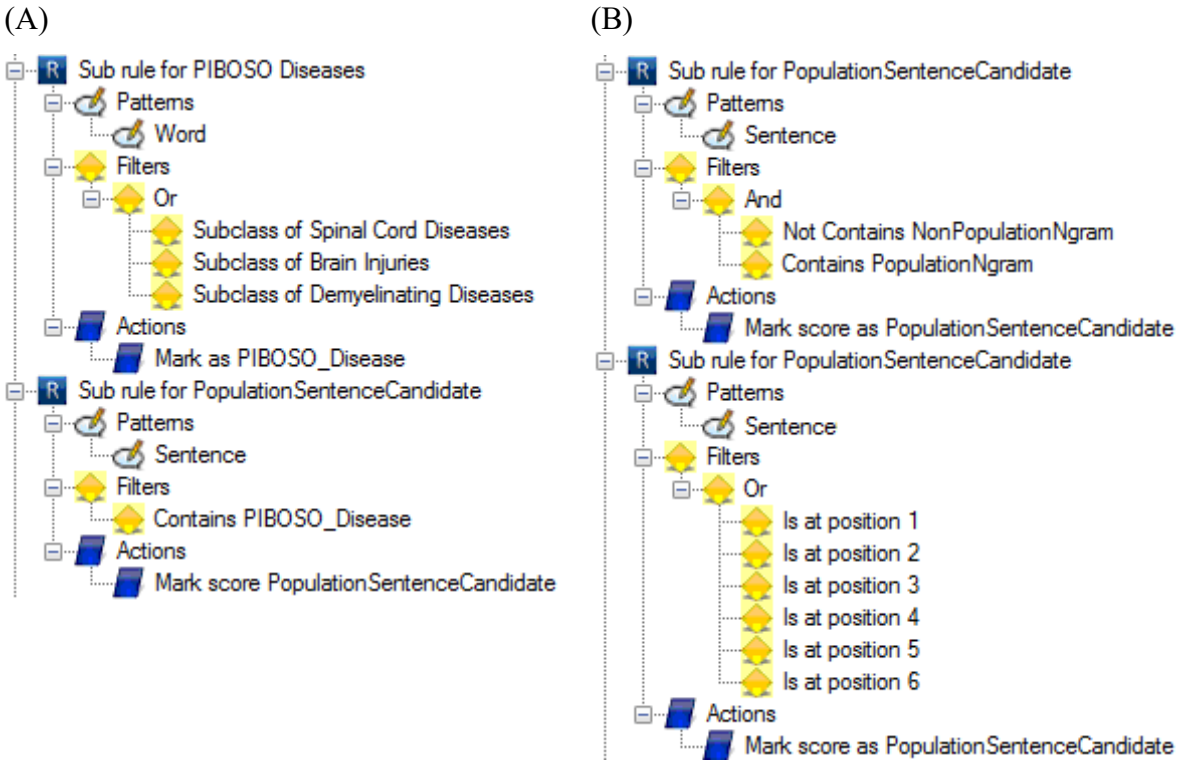


Figure 5.14: ARDAKE rules for PIBOSO Population sentence candidate annotations

The second rule in Figure 5.14 (A) assign a score to any sentence containing one or more “PIBOSO Diseases” annotation to be considered as a candidate for a Population sentence.

The first “PopulationSentenceCandidate” rule in Figure 5.14 (B) shows how negation and Boolean operators can be used to exclude undesired tokens. This rule assigns a score to sentences containing one or more population n-grams and no negative population n-grams. The second rule in the same figure increases the score of “PopulationSentenceCandidate” sentences if they are among the first 6 sentences in the document being analysed.

When a rule element is added to a rule, its properties can be set using the context-sensitive properties grid. Figure 5.15 shows the properties grids for three elements in rules defined in

Figure 5.14. Figure 5.15 (A) shows how to set the number of desired occurrences of the Word pattern in the "PIBOSO Diseases" rule. Here we indicate that any sequence of 1 to 5 consecutive words will be considered by the rule. Figure 5.15 (B) is the properties grid for the "Subclass of Brain Injuries" condition of the "PIBOSO Diseases" rule. The user does not have to type anything but rather selects the ontology and the parent concepts for creating annotations. The user can also use the Recursive property to specify whether or not to consider concepts that indirectly inherits from the selected parent concept. The "Subclass" condition is described in more details in section 5.6.2. Figure 5.15 (C) is for the "Mark score" action of the "PopulationSentenceCandidate" rule in Figure 5.14 (A) and it shows how a user can assign or increment the score for tokens matching the patterns part of rule and fulfilling its conditions. Note that the score can be decreased by specifying a negative value.

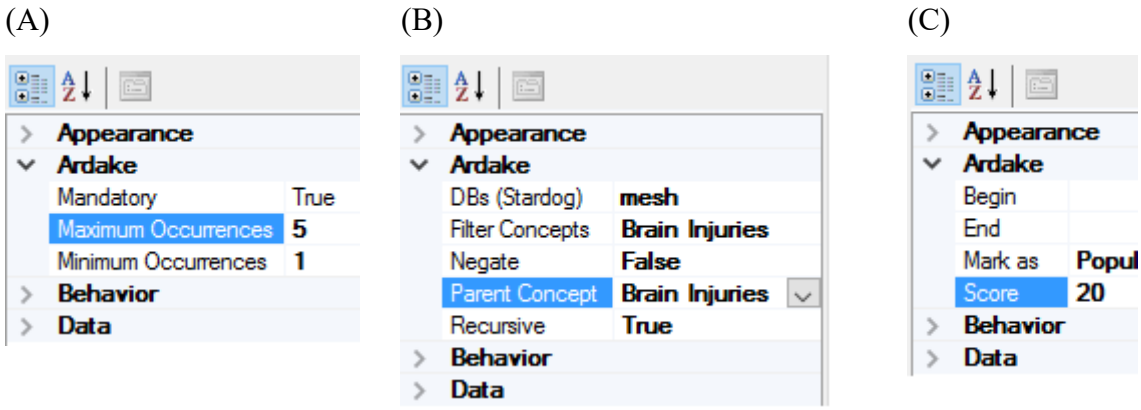


Figure 5.15: Properties grids for rules in Figure 5.14

Table 5.2 shows the Ruta script generated by ARDAKE for the rules in Figure 5.14.

Table 5.2: Ruta script generated by ARDAKE for rules in Figure 5.14

(A)	DECLARE PIBOSO_Disease; (W | (W W) | (W W W) | (W W W W) | (W W W W W)){OR(SubClassOf("<http://bioonto.de/mesh.owl#C10.228.854>", "mesh", true), SubClassOf("<http://bioonto.de/mesh.owl#C10.228.140.199>", "mesh", true), SubClassOf("<http://bioonto.de/mesh.owl#C10.314>", "mesh", true)

	) ->MARK(PIBOSO_Disease)}; Sentence{CONTAINS(PIBOSO_Disease)->MARKSCORE(20, PopulationSentenceCandidate)};
(B)	DECLARE PopulationSentenceCandidate; Sentence{AND(-CONTAINS(NonPopulationNgram), CONTAINS(PopulationNgram))->MARKSCORE(40, PopulationSentenceCandidate)}; Sentence {OR(POSITION(Document, 1), POSITION(Document, 2), POSITION(Document, 3), POSITION(Document, 4), POSITION(Document, 5), POSITION(Document, 6)) -> MARKSCORE(10, PopulationSentenceCandidate)};

5.6 Interfaces and Extensions to the UIMA Ruta Language

A powerful feature of the Ruta language that is reflected in ARDAKE is the ability to extend the rule language by adding definitions for new patterns, conditions, and actions without altering the core language. This is done by extending the ARDAKE node patterns, conditions, and actions classes as shown in Figure 5.16.

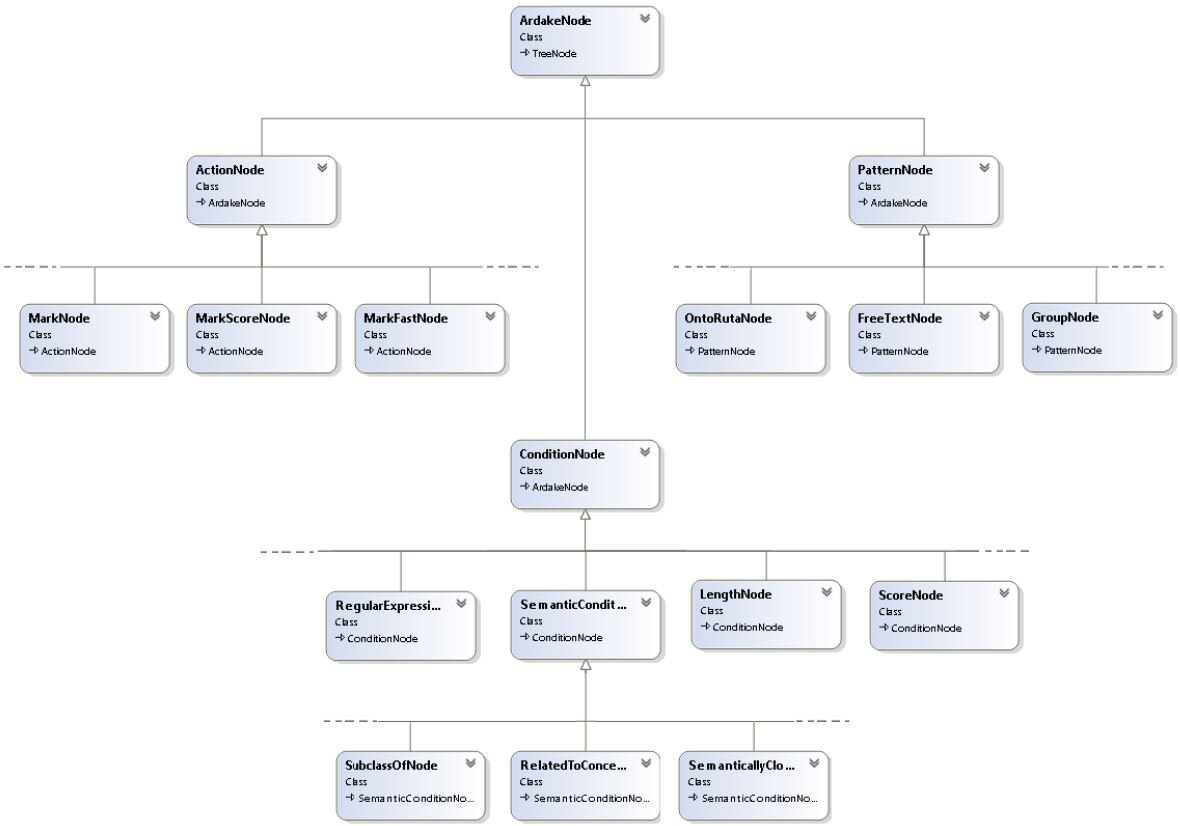


Figure 5.16: The main rule related classes in ARDAKE

In order to compensate for the lack of semantic rules in Ruta, most ARDAKE extensions we built for patterns, conditions, and actions are semantic-based. The following sections describe some of these extensions.

5.6.1 Pattern Extensions

Many domain experts prefer defining domain types and structures (concepts and relationships) formally using RDF or OWL ontologies. New ontologies and concepts are not created from scratch but are rather based on existing ones. An upper ontology defines the domain's main concepts and their relationships at a high abstraction level and helps regulating middle and lower level ontologies. For example, an upper ontology can define concepts "Product" and "Category" with their attributes and a mandatory relationship "exists in" that links "Product"

to “Category”. A domain ontology that defines concept “Milk” as a subclass of “Product” must link “Milk” to a subclass of “Category” such as “Dairy Products”.

ARDAKE provides an upper ontology (Figure 5.17) so that concepts and structures defined in domain ontologies can be used as patterns in annotation rules. This is done by linking concepts in domain ontologies to those in the ARDAKE upper ontology. The ARDAKE upper annotation ontology allows defining complex compound types and structures with any level of nested elements. The upper ontology also allows applying restrictions to concept defined in the domain ontology. For example, one can specify that the City pattern or element must appear after the Street name element in an Address concept/structure. Structures defined in domain ontologies can then be used in ARDAKE rules as shown in Figure 5.19.

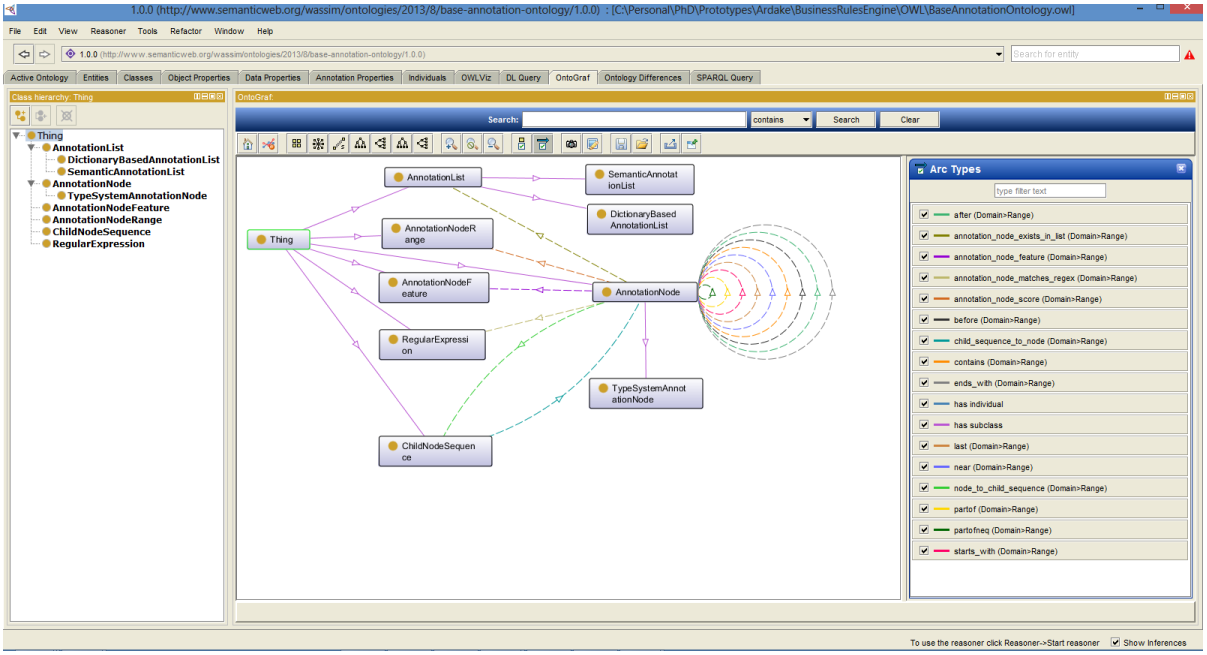


Figure 5.17: The ARDAKE annotation upper ontology

Figure 5.17 shows the structure of the ARDAKE annotation upper ontology whose concepts are explained in

Table 5.3.

Table 5.3: The ARDAKE Annotation Upper Ontology

Concept	Description
AnnotationNode	The main concept in the ARDAKE annotation upper ontology. It represents a pattern element in an ARDAKE rule.
TypeSystemAnnotationNode	This is a subclass of the AnnotationNode concept that represents a UIMA type system defined in an external UIMA Analysis Engine descriptor.
AnnotationNodeFeature	Represents a feature (property) for an annotation node. A feature is defined by its name and value (ex. “age”, “42”).
AnnotationNodeRange	Determines the minimum and maximum frequency for a pattern to be considered by a rule.
RegularExpression	Holds the regular expression for an ARDAKE rule condition of type “Matches regular expression”.
AnnotationList	Represents a collection of AnnotationNode.
DictionaryBasedAnnotationList	A subclass of AnnotionList to model annotations created based on a list using the ARDAKE MarkFast action.
SemanticAnnotationList	Another subclass of AnnotationList to model annotations created for concepts in ontologies identified using the ARDAKE “Related to” condition.
ChildNodeSequence	Acts as an intermediate layer between a parent annotation node and its children. It determines the order in which the children of a given parent element must appear in the rule.

Table 5.4 shows some object properties in the ARDAKE annotation upper ontology along with their domains, ranges, and equivalent ARDAKE elements.

Table 5.4: The Object Properties in the ARDAKE Annotation Upper Ontology

Object Property	**Domain**
	Range
	Equivalent ARDAKE Element
annotation_node_exists_in_list	AnnotationNode
	AnnotationList
	"Is in list" condition
annotation_node_feature	AnnotationNode
	AnnotationNodeFeature
	"Create annotation with features" action
annotation_node_matches_regex	AnnotationNode
	RegularExpression
	"Matches regular expression" condition
annotation_node_score	AnnotationNode
	AnnotaionNodeRange
	"Mark score" action
child_sequence_to_node	ChildSequenceNode
	AnnotaionNode
	"Is in list" condition
node_to_child_sequence	AnnotaionNode
	ChildSequenceNode
	"Is in list" condition

The remaining object properties in the ARDAKE annotation upper ontology have AnnotationNode as their Domain and Range and translate into identical ARDAKE conditions. These properties are "after", "before", "contains", "ends_with", "last", "near", "partof", "partofneq", "precedes", and "starts_with".

In addition to object properties, the ARDAKE annotation upper ontology has a number of data properties. An interesting data property for the AnnotationNode concept is the "is_root"

property that identifies the starting (root) concepts while converting structures defined in the ontology into annotation rules. Another useful data property for the AnnotationNode concept is the "is_mandatory" that determines if a concept is mandatory or optional for a structure. This maps into the "Mandatory" property of an ARDAKE pattern.

The ChildNodeSequence concept acts as an intermediate layer between a parent annotation node and its children. It determines the order in which the children of a given parent element must appear in the rule. ChildNodeSequence can be ignored if the order of children is not important.

Figure 5.18 shows the "before" relationship connecting different concepts such as AdressLine1 to AddressLine2, AddressLine2 to PostalCode, Line1Token1 to Line1Token2, and Line2Token1 to Line2Token2.

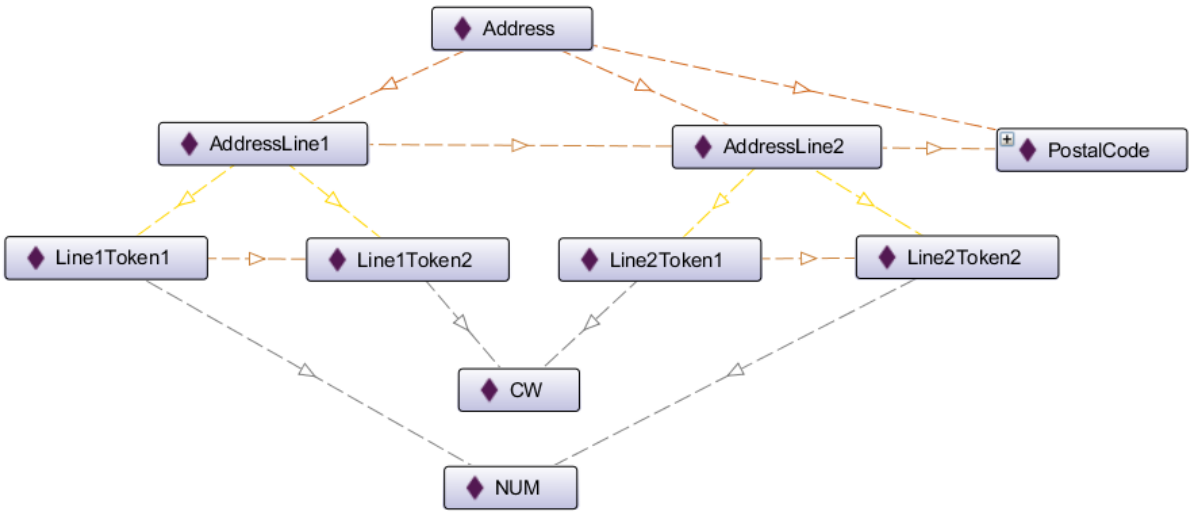


Figure 5.18: The Address ontology

Line1Token1, Line1Token2, Line2Token1, and Line2Token2 are all instances of type ChildNodeSequence concept in the ARDAKE upper ontology. Instances of this concept help determining the order of child concepts but do not appear in resulting annotation rules. Table

5.5 shows the Ruta script that ARDAKE generates for the Address ontology defined in Figure 5.18.

Table 5.5: ARDAKE generated Ruta script for the Address ontology

```
DECLARE AddressLine1;
 NUM CW { -> MARK(AddressLine1, 1, 2)};
DECLARE AddressLine2;
 CW NUM { -> MARK(AddressLine2, 1, 2)};
DECLARE Address;
 AddressLine1 AddressLine2 PostalCode { -> MARK(Address, 1, 3)};
```

To add an Ontology-based pattern to an ARDAKE rule, simply drag the "Ontology Pattern" into the patterns node of the rule and set its properties as shown in Figure 5.19. The ontology can be loaded from an RDF/OWL file or directly from the web using its URL. It is also possible to load ontologies from triple stores such as a Stardog data store.

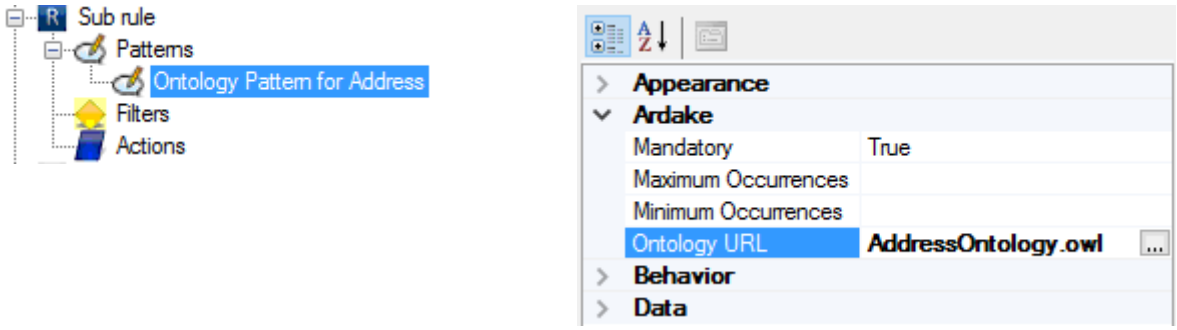


Figure 5.19: Ontology pattern for Address

ARDAKE generates annotation rules for "Ontology" patterns linked to OWL/RDF ontologies. Similar extensions can be added to create pattern extensions for relational databases, UML, and .net classes using reflection.

5.6.2 Condition Extensions

A simple but useful condition extension in ARDAKE is the length extension that evaluates to true for tokens whose length is within the minimum and maximum length properties of the condition. The length condition played an important role in improving the F-Score for the Population and Intervention annotations in the PIBOSO corpus. As shown in section 7.4 in CHAPTER 7, the length of sentences containing population and intervention patterns in the PIBOSO corpus fall within a specific range.

"Subclass of" is a semantic condition extension that can be used to identify tokens matching any concept that directly or indirectly inherit from a given concept in an ontology. After adding a "Subclass of" condition to the conditions node of a rule, a user can use the properties of the condition to select an ontology from a Stardog triple store. The user can then filter and search concepts within the ontology to select the root concept for the condition. The user can finally set the "Recursive" property of the rule to determine whether or not to consider concepts that indirectly inherit from the root one. The "Subclass of" condition also played a key role in improving the F-Score for the Population and Intervention annotations in the PIBOSO corpus.

Other semantic and NLP condition extensions can also be added to test NLP properties or to check whether or not tokens are semantically related or semantically close. This can be done using the Semantic Measures Library & ToolKit [45; 46] available at (http://www.semantic-measures-library.org/sml/index.php) or similar libraries.

5.6.3 Action Extensions

ARDAKE allows growing (enriching) knowledge bases with concepts and relationships learned while analyzing documents. This is enabled by a number of semantic action extensions that include "AddConcept", "AddObjectProperty", "AddDataProperty", "AddInstance", "RemoveConcept", "RemoveObjectProperty", "RemoveDataProperty", and "RemoveInstance".

One can decide to do a double-pass corpus analysis where an ontology is created or enriched with concepts and relationships learned during the first pass. This ontology can then be used with other data sources during the second pass to look for specific patterns/information within the corpus.

5.7 Chapter Summary

Using the right tools and following a suitable methodology are key success factors for large and complex projects. The right tools are the ones that make the most challenging tasks simple or significantly simpler. A good methodology serves as a roadmap in long running projects where it is easy to lose track. Most KEFUD projects are both complex and long requiring a rich set of powerful, easy to use, tools as well as a clear and easy to follow methodology.

In this chapter, we presented our prototype, called ARDAKE, that includes a number of powerful, yet easy to use, tools that we developed to help KEFUD miners with their most challenging tasks at various stages of a KEFUD project. We defined the requirements of our prototype in Section 5.2 then showed, in subsequent sections, how all requirements were met. Table 5.6 summarizes the ARDAKE requirements and indicates where each one is fulfilled.

Table 5.6: ARDAKE Requirements

Requirement	**Implemented in**	**Section(s)**
1) Non-technical domain experts should be able to create KE rules using a visual, user friendly, rule editor without having to write any code.	ARDAKE Rules Composer	5.3.5
2) Users should have access to a set of simple integrated tools to help them with the ETL and the analysis of textual corpora.	The Corpus Extractor The Corpus Transformer The Corpus Loader The Corpus Analyser	5.3.1 5.3.2 5.3.3 5.3.4

3) Users should be able to visualize and analyse the rules execution results in order to have a better insight on how rules can be optimized.	The Rules Results Analyser	5.3.7
4) ARDAKE should assist users through the automatic generation of n-gram inclusion and exclusion rules as described in 4.5.2.	The Corpus Analyser	5.3.4
5) To maintain the efficiency of rule-based KE, ARDAKE must support rules of various types including linguistic, statistical, and semantic rules. All ARDAKE rules will be created in a consistent way using a simple visual rule editor.	ARDAKE Rules Composer Rules structures Ruta Rules Extensions	5.3.5 5.5.1 5.6
6) ARDAKE will assist users finding the combination of rules that produces the best results based on F-Score.	The Rules Combiner	5.3.8
7) It should be possible to share rules with other ARDAKE users and use existing rules to define more complex ones.	ARDAKE Rules Composer Rules Examples	5.3.5 5.5.3
8) Users should be able to run rules directly from ARDAKE or generate equivalent Ruta rules that can be used in any UIMA Ruta compatible system.	ARDAKE Rules Composer The Ruta Generator	5.3.5 5.3.6

CHAPTER 6

Corpus Preprocessing

6.1 Chapter Overview

The Extract-Transform-Load (ETL) process is an expensive but essential part of most knowledge extraction projects. Designing and implementing ETL takes up to 70% of the time and cost resources [47]. This high ETL cost was reported for projects dealing with structured data. Since it is harder to work with unstructured data, it is expected that the cost of ETL from unstructured data sources is even higher. It is therefore crucial to develop and rely on advanced specialized tools to simplify the ETL process when working with unstructured data.

In this chapter, we present the tools we used and the steps we took to find a good corpus and preprocess it before loading into our database that is designed for visualization, easy and efficient, data and results analysis.

6.2 Finding the Right Corpus

The efficiency and performance of Knowledge Extraction (KE) solutions and tools cannot be measured and evaluated without running them over a corpus that represents the overall data of the respective domain. The corpus must be approved and manually annotated by domain experts so that results produced by KE solutions and tools can be compared against those created by the domain experts. Those solutions and tools are then evaluated by how close the results they produce match the ones done by domain experts. Solutions and tools producing the best results can be used as a benchmark by future solutions and tools that try to beat their scores. Finding the right corpus to create, train and evaluate KE models is therefore an essential step for any KE solution.

After evaluating annotated corpora for different domains including financial, sports, and medical, we found that the most suitable corpus to evaluate our solution is the NICTA-PIBOSO corpus described in section 6.4.

While expert-level knowledge is not required to evaluate the performance of KE solutions when run against an already annotated corpus, basic knowledge is necessary to compare the results and make the right judgments. In the next sections, we introduce EBM and its related PICO and PIBOSO frameworks.

6.3 EBM PICO

Evidence Based Medicine (EBM) is an approach to better teaching medical and clinical practices by making the search for answers in large volume medical libraries more formal and systematic [48; 49].

Keywords search without a formal context usually lead to a large number of unrelated results. For example, a keyword search for asthma will return all articles containing the word asthma or related synonyms in their title, abstract, or text. This does not take into consideration the patient's gender, age, or other criteria such as what kind of intervention is needed and the desired outcome. Even if all those elements are specified in a keyword search string, different people would specify them in different ways and orders making it hard for the search engine to identify them and use them for a more precise search result.

With EBM, medical students and practitioners perform more contextual searches by specifying keywords in different categories. EBM recommends that physicians express their questions/queries in terms of the Problem/Patient/Population, Intervention, Comparison, and Outcome or PICO in short.

PICO is not the only EBM framework for formulating clinical questions. Other PICO variations such as PECORD and PIBOSO are also used. Different studies showed that, despite their limitations, PICO and similar frameworks help improving query results when searching for answers to medical questions in already annotated abstracts.

What limits the success of PICO and similar frameworks is the fact that a large number of medical questions do not necessarily have all the elements of the framework. In fact, many questions will only have 1 or 2 elements of the underlying framework but even in this case, the frameworks still present a good value in assisting physicians practicing EBM. One of the biggest challenges of EBM is the annotation of PICO elements in medical abstracts and text [50].

6.4 NICTA-PIBOSO Corpus

The Australasian Language Technology Association (ALTA) organizes annual events and workshops about language technologies. ALTA's goal is to enable computers to better process human languages and to allow more sophisticated access to stored information.

In 2012, ALTA organized a competition to build automatic sentence classifiers that can map the content of sentences from biomedical abstracts into a set of pre-defined EBM categories. The competition was sponsored by the National Information Communications Technology Australia (NICTA) which is one of the largest Australian research center dedicated to Information Communication Technology research. NICTA also provided the annotated corpus and related data sets for the competition. The annotations were based on an EBM PICO variation called PIBOSO that drops the C (Comparison) element of PICO and adds three new elements B (Background), S (Study Design), and O (Other). Here is a short description of each element in the PIBOSO schema:

- **Population**: The group of individual persons, objects, or items comprising the study's sample, or from which the sample was taken for statistical measurement;
- **Intervention**: The act of interfering with a condition to modify it or with a process to change its course (includes prevention);
- **Background**: Material that informs and may place the current study in perspective, e.g. work that preceded the current; information about disease prevalence; etc.;
- **Outcome**: The sentence(s) that best summarize(s) the consequences of an intervention;
- **Study Design**: The type of study that is described in the abstract;
- **Other**: Any sentence not falling into one of the other categories and presumed to provide little help with clinical decision making, i.e. non-key or irrelevant sentences.

The NICTA-PIBOSO Corpus has a total of 1000 expert-annotated structured (i.e. includes heading such as Method, Results, Conclusions, etc.…) and unstructured abstracts where 800 are used for training and 200 for testing. Unlike structured abstracts, unstructured abstracts do not include any headings. The 800/200 of training/testing abstracts was decided by the creator or the NICTA-PIBOSO corpus and was used by those who participated to the ALTA-NICTA PIBOSO contest.

6.5 Downloading and Preparing abstracts for the PIBOSO corpus

The NICTA-PIBOSO corpus data is available on Kaggle at https://inclass.kaggle.com/c/alta-nicta-challenge2/data. The following files can be downloaded from there:

Train.csv: a file containing comma separated values used to train models on labeling sentences using the PIBOSO schema. Sentences in this file were taken from 800 PUBMED abstracts. Each line in this file contains five fields.

- Prediction: a 0 or 1 value that indicate whether the sentence of the line matches the PIBOSO label (class) of the same line.
- Label: one of the PIBOSO labels (Population, Intervention, Background, Outcome, Study design, or Other).
- Document: the PUBMED identifier of the source abstract from which sentences were taken. Appending this number to the main PUBMED URL "http://www.ncbi.nlm.nih.gov/pubmed/" on the National Library of Medicine website creates a link to the actual abstract text. For example, the link to obtain the text of the abstract with the document number 10072623 is "http://www.ncbi.nlm.nih.gov/pubmed/10072633". This gives access to the abstract in html format. To obtain abstracts in XML format, add "?dopt=XML" at the end of the URL after the document number.
- Sentence: The sequence number of the sentence within the source abstract text.
- Text: the text of the sentence of the current line.

Test.csv: this is used to test the performance of classification models. It contains the same fields as Train.csv except the Prediction column that has to be provided by classifiers. Sentences in this file were taken from 200 PUBMED abstracts that are in mutual disjunction with the train.csv ones. Related abstracts can be downloaded in html or XML format using the same method described above to download training abstracts.

More files are provided for training purposes such as files containing sentences of the training abstracts with NLP tags as well as other files that contain the headers in each structured abstract.

ARDAKE rules are converted into UIMA Ruta scripts that gets included into a UIMA Analysis Engine then executed using the Eclipse-based UIMA Document Analyser. The UIMA Document Analyser runs UIMA Analysis Engines over a set of files in an input folder. Therefore, full abstract text for the PIBOSO corpus has to be stored in files in order to annotate them using annotation rules created with ARDAKE. The ARDAKE Corpus Extractor was used to download all training and test abstracts by parsing the csv files of the PIBOSO corpus and looping through the document numbers. For each document number, the URL of the abstract in XML format was constructed using the method described above.

6.6 Importing Abstracts and Sentences into the ARDAKE database

The training and test CSV files described in section 6.5 contain, for each sentence, the sentence text, its sequence number within its parent abstract, as well as the PUBMED document ids of the abstract. Therefore, they can be used to import the PIBOSO abstracts and sentences into the ARDAKE database. The PIBOSO corpus also has, for each structured abstract, a text file that has the PUBMED abstract ID as a name and that contains the list of headers along with their locations within the abstract. This can be used to set the optional IsStructured field of the Documents table in the ARDAKE database.

Before importing the PIBOSO abstracts, a new row was inserted in the Corpora table for the PIBOSO corpus. The corpus id field is an auto-generated number; the corpus name was set to “PIBOSO”; and a description of the PIBOSO corpus was provided in the CorpusDescription field of the Corpora table.

For each abstract in the PIBOSO corpus, a new row was inserted in the Documents table of the ARDAKE database. The document id is an auto-generated number; the corpus id is the PIBOSO corpus id as defined in the Corpora table; the document type is set to “test” for

documents in the Test.csv and to "training" for documents in Training.csv respectively. Document contents can be obtained either by downloading them as described in section 6.5 or by concatenating sentences, with the same document id, in the right order based on their sequence number in the csv file. The data in the csv files are enough to populate the Sentences table as these contain all fields in the Sentences table, namely the document (abstract) id, the sentence sequence number within the abstract, and the sentence text. Note that the sentence sequence number is unique within a given abstract as no two different sentences can have the same sequence number in the same abstract. This means that any sentence in the ARDAKE database can be uniquely identified by its document id and sequence number.

6.7 Importing the PIBOSO Manual Annotations into the ARDAKE database

The PIBOSO corpus includes a text file called gs.txt that has the list of all manual annotations for the training and test abstracts. Each line in gs.txt contains three tab separated fields that are the abstract document id, the sentence sequence number within the abstract, and the manual annotation. A sentence that has multiple manual annotations has multiple lines in gs.txt (one line per annotation) as shown for sentence 1 in document 10850747 in the following snippet taken from gs.txt.

Table 6.1: PIBOSO Annotations Counts

```
10847225  21   other
10847225  22   outcome
10847225  23   outcome
10850747  1    background
10850747  1    population
10850747  1    intervention
10850747  2    outcome
```

In order to import manual annotations, a test run has to be created first. This is done by inserting a new row in the TestRuns table. For PIBOSO, the test run date was set to the date on which the manual annotations were imported into the ARDAKE database, the ARDAKE file content and Ruta script fields were left empty, and the test run description was set to "Manual annotations for the PIBOSO corpus". The PIBOSO manual annotations were imported into the ARDAKE database by reading the content of gs.txt line by line and inserting a new row in

XRef_Sentences_Annotations for each line. For each row, the test run id is the id of the manual test run as defined in the TestRuns table, the annotation type is obtained from the last element in each line read from gs.txt. To get the sentence id, the first two elements in each line in gs.txt (document id and sentence sequence number) are read then used to query the Sentences table for the uniquely matching sentence id.

6.8 Importing ARDAKE/UIMA Annotations into the ARDAKE database

ARDAKE rules can be executed directly from the ARDAKE menu or used to generate Ruta scripts that are executed in Eclipse using the UIMA Document Analyser that creates an xml document with the xmi extension in which it stores all annotations for each document being analysed. In order to reuse the existing functionality of importing manual annotations into the ARDAKE database, UIMA annotations in xmi files are used to generate a file that has the same structure as gs.txt. This file is then read and imported the same way gs.txt is imported into the ARDAKE database.

6.9 Chapter Summary

We described the tools we used and the steps we took to find a good corpus and to preprocess it before loading it into our database for further analysis. This is a crucial step given its complexity and its big influence on all subsequent steps in any KEFUD project. Meta-data and results extracted from unstructured data sources and their training and test sets respectively should be stored in a manner that makes it simple to analyse and visualize. As we show in the following chapter, our database is designed to store meta-data and results in a way that greatly simplifies visualization and data analysis. More importantly, our database and tools allow the automatic generation of more efficient KEFUD rules based on existing ones.

CHAPTER 7

Data Analysis

7.1 Chapter Overview

KE from any domain requires some level of understanding of that domain as well as the corpus and data from which the knowledge is to be extracted. The minimum required level of understanding depends on the degree to which we rely on automatic parsing and rules/models generation. Unsupervised ML algorithms require no, or minimal, understanding of the underlying domain and data. However, these algorithms are often based on statistical analysis only and are therefore limited and lack the semantic and business aspects in the rules and models they produce. This is particularly true for unstructured data where NLP and TDF are used first to create structured data before running statistical-based unsupervised algorithms to generate rules and models.

Defining KE rules manually requires a high level of understanding of the domain and corpus in question whether or not training data is available. This is why domain experts are sometimes the best ones to define KE rules. Domain experts know the business rules featuring the domain knowledge but still need tools to help them discover and understand the structural, linguistic, statistical, and semantic characteristics of the knowledge to extract.

7.2 Studying the PIBOSO Domain and Training Set

Before defining any annotation rules, one should have some basic understanding of the domain. When good training sets are available, they should also be studied, visualized, and analysed with the purpose of finding distinguishing properties for each annotation type. We studied the PICO/PIBOSO literature and training sets to gain basic knowledge on the topic so that we can do more in-depth analysis and start writing elementary annotation rules. Our focus was mainly on the Population and Intervention annotations since, so far, none of the existing algorithms/tools has succeeded to precisely identify them in the PIBOSO corpus. The literatures and the manual annotations helped identifying a number of common patterns (ex.

Age, gender, etc.) in Population and (treatments, drugs, procedures) in Intervention annotations. The MeSH ontology is another good source of information about patterns commonly found in PIBOSO and other medical corpora.

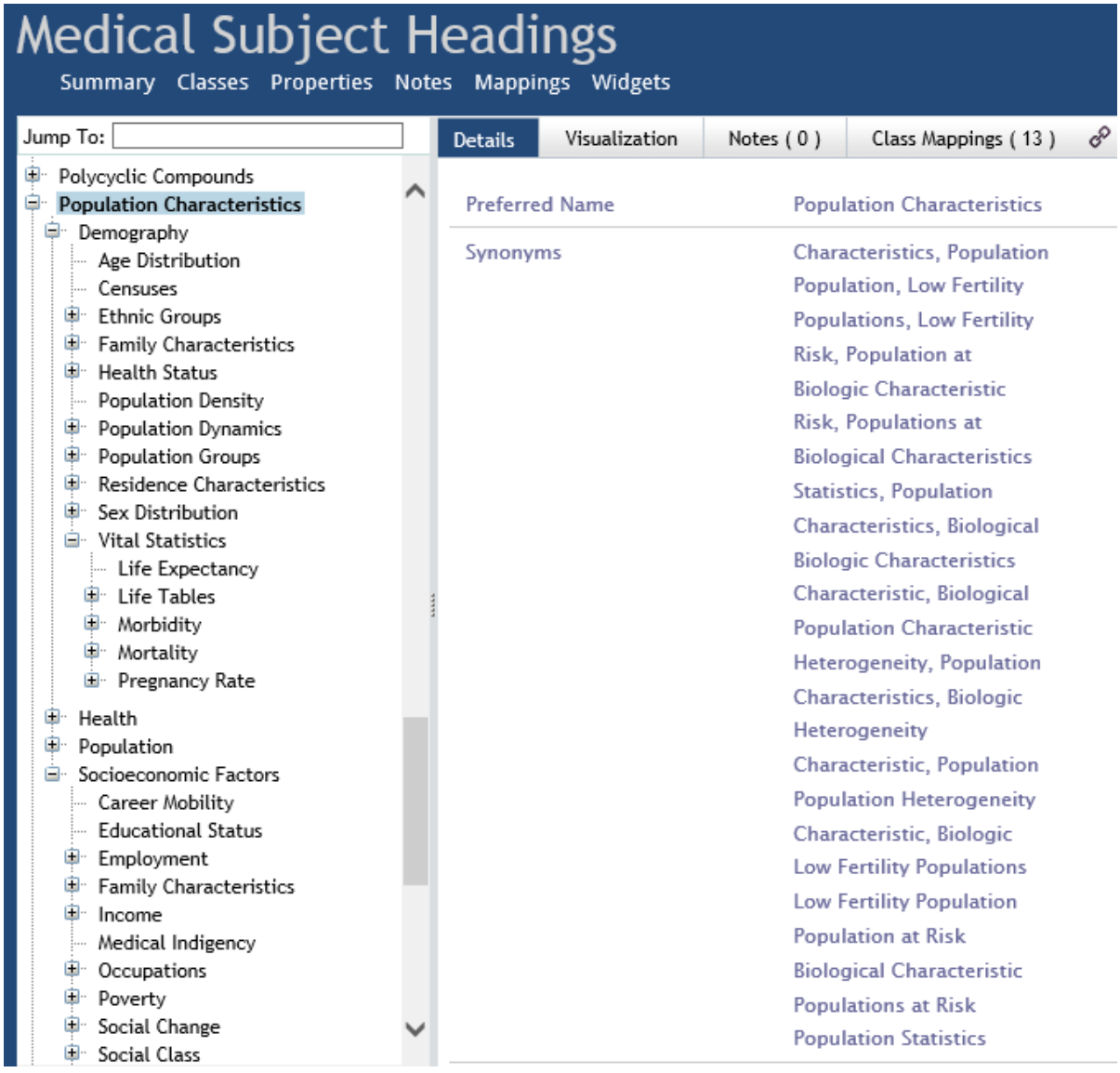


Figure 7.1: The Population Characteristics class and its subclasses in MeSH

Figure 7.1 shows the "Population Characteristics" class and some of its subclasses in the MeSH ontology. These definitions and their synonyms can be very useful for creating basic rules to identify Population annotations in medical abstracts. Other classes in MeSH such as "Surgical

Procedures, Operative" and "Therapeutics" can be used to create rules for the Intervention annotations. Ideally, a more specific ontology for the PIBOSO corpus should be used but since such ontology could not be found, we decided to use the MeSH ontology instead.

7.3 Using the ARDAKE Corpus Analyser

We used the ARDAKE Corpus analyser to create n-grams decision trees for the Population sentences in the training set of the PIBOSO corpus. Table 7.1 shows a subset of the results produced by the ARDAKE Corpus Analyser for creating the unigram decision tree with a minimum frequency of 3 and the minimum F-Score of 90%.

Table 7.1: Subset 1 of the ARDAKE Corpus Analyser results

```
  <name_635790719492055913 Text="PositiveNGrams">
    <name_635790719492055913 Text="crossover" PCount="4" NCount="0" />
    <name_635790719492055913 Text="July" PCount="3" NCount="0" />
    <name_635790719492055913 Text="Sprague-Dawley" PCount="3" NCount="0" />
    <name_635790719492055913 Text="corn" PCount="3" NCount="0" />
    <name_635790719492055913 Text="formula-fed" PCount="3" NCount="0" />
    <name_635790719492055913 Text="Indian" PCount="3" NCount="0" />
    <name_635790719492055913 Text="rabbits" PCount="3" NCount="0" />
    <name_635790719492055913 Text="cord-injured" PCount="3" NCount="0" />
  </name_635790719492055913>
  <name_635790719492055913 Text="NegativeNGrams">
    <name_635790719492055913 Text="significant" PCount="0" NCount="308" />
    <name_635790719492055913 Text="significantly" PCount="0" NCount="292" />
    <name_635790719492055913 Text="trials" PCount="0" NCount="245" />
    <name_635790719492055913 Text="therapy" PCount="0" NCount="241" />
    <name_635790719492055913 Text="increased" PCount="0" NCount="238" />
    <name_635790719492055913 Text="quality" PCount="0" NCount="205" />
    <name_635790719492055913 Text="evidence" PCount="0" NCount="184" />
...
  </name_635790719492055913>
  <name_635790719492212171 Text="OtherNGrams">
    <name_635790719493618523 Text="patients" PCount="135" NCount="1317">
      <name_635790719493618523 Text="PositiveNGrams">
```

```
        <name_635790719493618523 Text="retrospective" PCount="4" NCount="0" />
        <name_635790719493618523 Text="obstruction" PCount="4" NCount="0" />
        <name_635790719493618523 Text="recruited" PCount="3" NCount="0" />
        <name_635790719493618523 Text="skin" PCount="3" NCount="0" />
        <name_635790719493618523 Text="aged" PCount="3" NCount="0" />
        <name_635790719493618523 Text="clinic" PCount="3" NCount="0" />
        <name_635790719493618523 Text="Hospital" PCount="3" NCount="0" />
        <name_635790719493618523 Text="carried" PCount="3" NCount="0" />
        <name_635790719493618523 Text="determined" PCount="3" NCount="0" />
…
      </name_635790719493618523>
      <name_635790719493618523 Text="NegativeNGrams">
        <name_635790719493618523 Text="therapy" PCount="0" NCount="67" />
        <name_635790719493618523 Text="risk" PCount="0" NCount="50" />
        <name_635790719493618523 Text="symptoms" PCount="0" NCount="44" />
…
  </name_635790719492212171>
```

Table 7.1 shows that only few n-grams were repeated 3 or 4 times in the Population sentences and less than 3 times in the non-population sentences. On the other hand, many n-grams have high frequencies in the non-population sentences and frequencies of 2 or less in the population sentences. This means that only few keywords can be used in Population inclusion rules but many keywords can be used in exclusion rules as they are frequently used in non-population sentences but not in the population ones.

Table 7.1 also shows that many n-grams frequently appear in both population and non-population sentences. Having these n-grams in inclusion rules results in a large number of false positives and having them in the exclusion rules would eliminate many valid population sentences resulting in a large number of false negatives. Therefore, these n-grams must be used in conjunction with other n-grams to create inclusion and exclusion rules. For example, the word “patients” appears 135 times in population sentences and 1317 times in non-population sentences. However, “patients” co-occurs 4 times in same sentences with “retrospective”

and/or “obstruction” in population sentences only. Similarly, “patients” co-occurs 67, 50, and 45 times with “therapy”, “risk”, “symptoms” respectively in non-population sentences only. Changing the parameters of the ARDAKE Corpus Analyser yields to a different n-gram decision tree as shown in Table 7.2. This tree was produced for unigrams with a minimum frequency of 5, and the minimum F-Score of 90%.

Table 7.2: Subset 2 of the ARDAKE Corpus Analyser results

```
  <name_635792614000163346 Text="PositiveNGrams">
    <name_635792614000163346 Text="man" PCount="7" NCount="0" />
  </name_635792614000163346>
  <name_635792614000163346 Text="NegativeNGrams">
    <name_635792614000163346 Text="significant" PCount="0" NCount="308" />
    <name_635792614000163346 Text="significantly" PCount="0" NCount="292" />
    <name_635792614000163346 Text="trials" PCount="0" NCount="245" />
    <name_635792614000163346 Text="therapy" PCount="0" NCount="241" />
    <name_635792614000163346 Text="increased" PCount="0" NCount="238" />
…
  </name_635792614000163346>
  <name_635792614000243403 Text="OtherNGrams">
    <name_635792614000243403 Text="patients" PCount="135" NCount="1316">
      <name_635792614000243403 Text="PositiveNGrams">
        <name_635792614000243403 Text="prospective" PCount="7" NCount="0" />
        <name_635792614000243403 Text="conducted" PCount="5" NCount="0" />
        <name_635792614000243403 Text="referred" PCount="5" NCount="0" />
      </name_635792614000243403>
      <name_635792614000243403 Text="NegativeNGrams">
        <name_635792614000243403 Text="group" PCount="0" NCount="80" />
        <name_635792614000243403 Text="injury" PCount="0" NCount="69" />
        <name_635792614000243403 Text="care" PCount="0" NCount="68" />
        <name_635792614000243403 Text="risk" PCount="0" NCount="50" />
        <name_635792614000243403 Text="outcome" PCount="0" NCount="48" />
        <name_635792614000243403 Text="symptoms" PCount="0" NCount="44" />
…
```

By observing the results in Table 7.2, more n-gram co-occurrence inclusion and exclusion rules can be created.

7.4 Visualizing and Analyzing the PIBOSO Training Set

One of the advantages of storing corpora (especially training sets) in a relational database is to enable the easy visualization and analysis of data using built-in and custom database functionalities as well as existing visualization tools such as Tableau and QlikView. This helps getting a quick insight into the manual annotations and allow building basic inclusion and exclusion annotation rules with pretty high precisions. Data mining tools can also be used to get advanced statistics about the manual annotations.

We used many built-in and custom functionalities and a number of visualization tools to get distinguishing factors for the PIBOSO annotations. All tools led to the same conclusions showing strong correlations between some sentence annotation types and the lengths and the positions of those sentences within their abstracts as shown in the screenshots below for different visualizations created using the Tableau software.

Figure 7.2 shows the number of annotations for each annotation type in structured and unstructured abstracts of the PIBOSO corpus. While this can be easily produced at the database level and have another tool such as Microsoft Excel produce the graphics, visualization tools such as Tableau and Qlik do the work with few mouse drag/drops. The same data can be shown in different visual formats for easy interpretation.

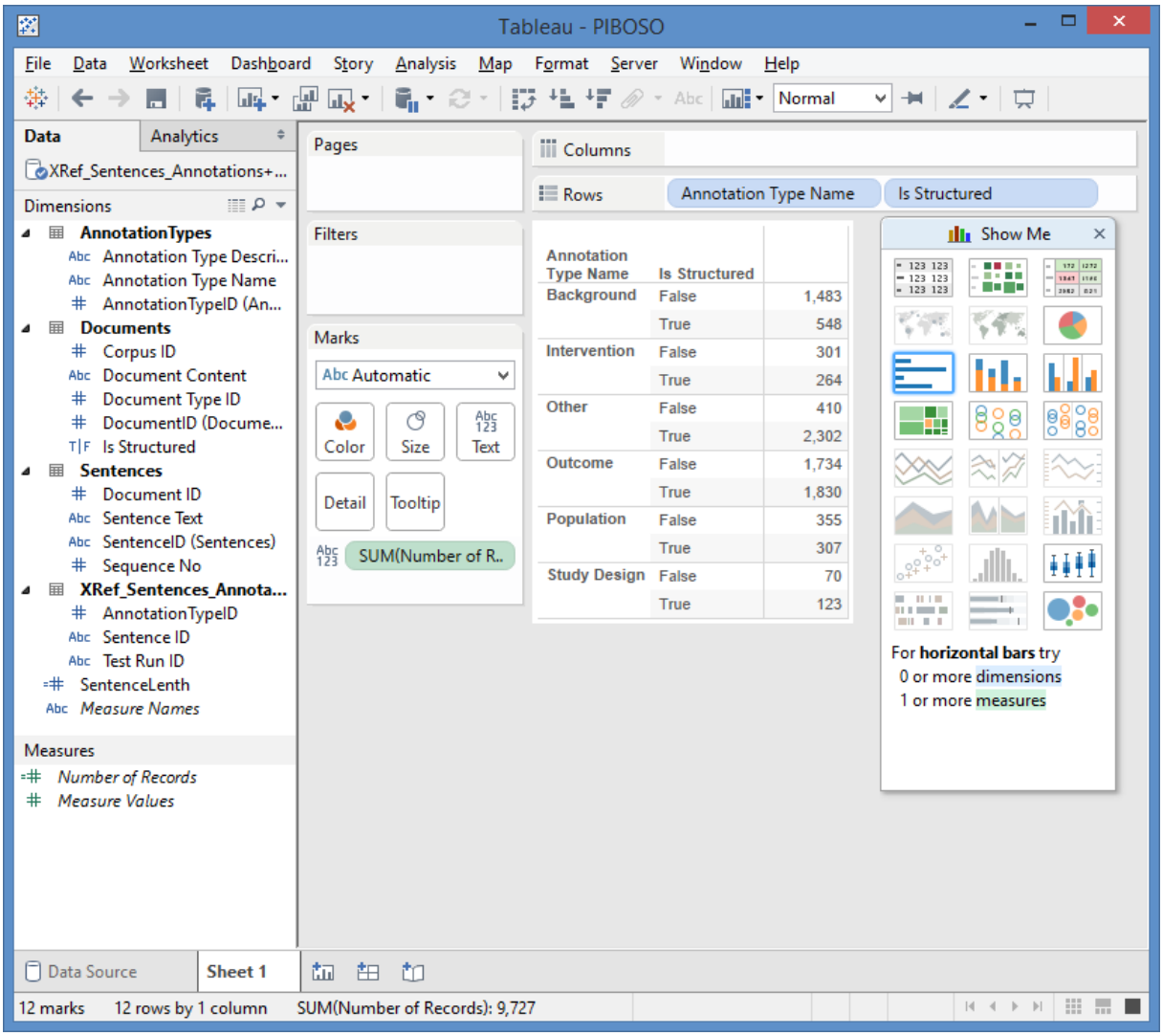


Figure 7.2: PIBOSO annotations counts in structured and unstructured abstracts

Figure 7.3 shows the same data presented in Figure 7.2 using a stack chart. By looking at the chart in Figure 7.3, one can immediately realize that the PIBOSO corpus has more Outcome annotations than any other annotation types. It is also clear that Outcome annotations are evenly distributed between structured and unstructured abstracts. This is not the case for the "Other" annotation type where the number of annotations in structured abstract is much larger than those in the unstructured ones. The chart also shows that three quarters of the "Background" annotations are in unstructured abstracts. Another obvious information that can be obtained from the chart is that there are not as many annotations for "Study Design",

"Intervention", and "Population". Some researchers use this later information as a justification for not having high F-Scores for the "Population" and "Intervention" annotations.

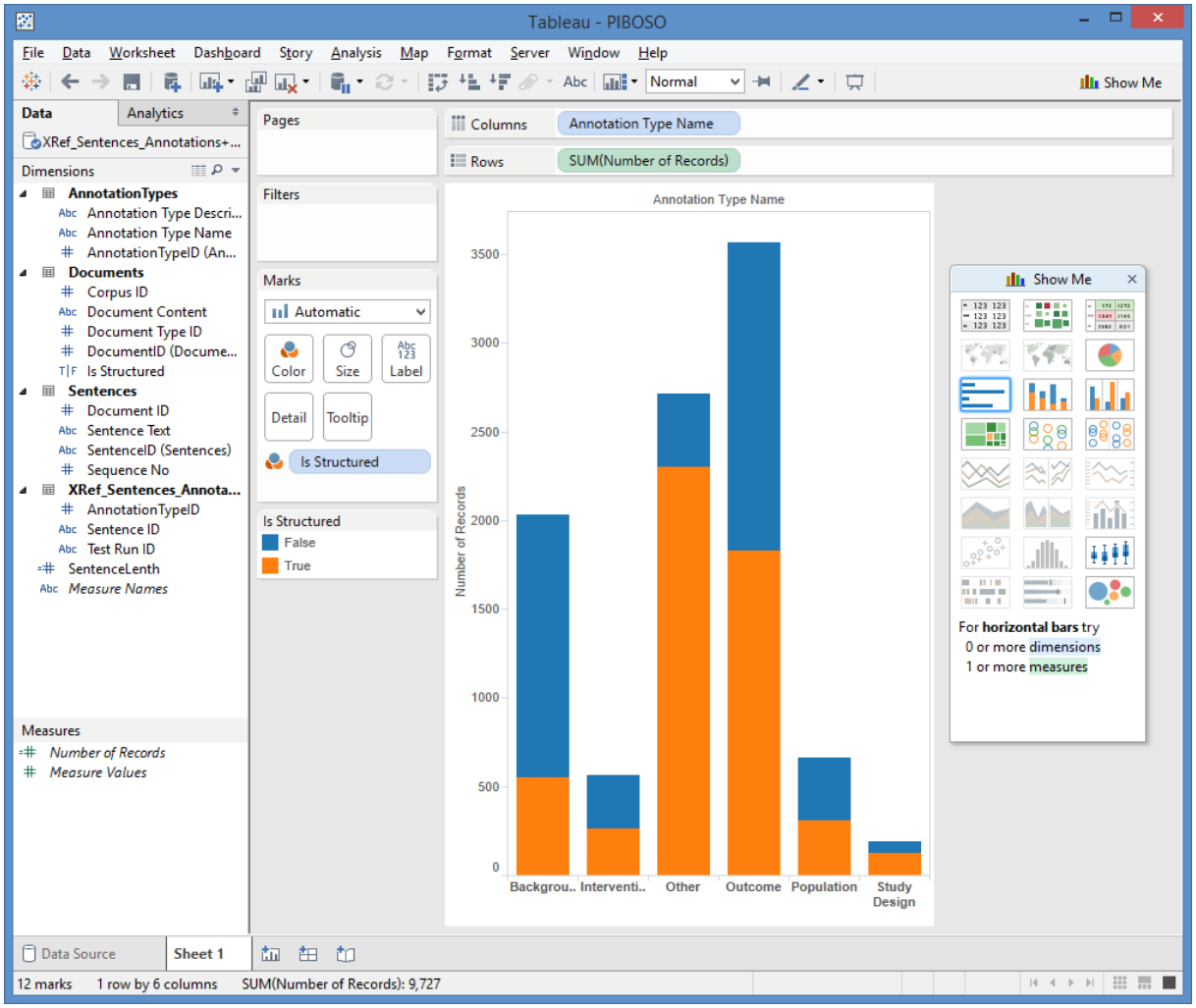
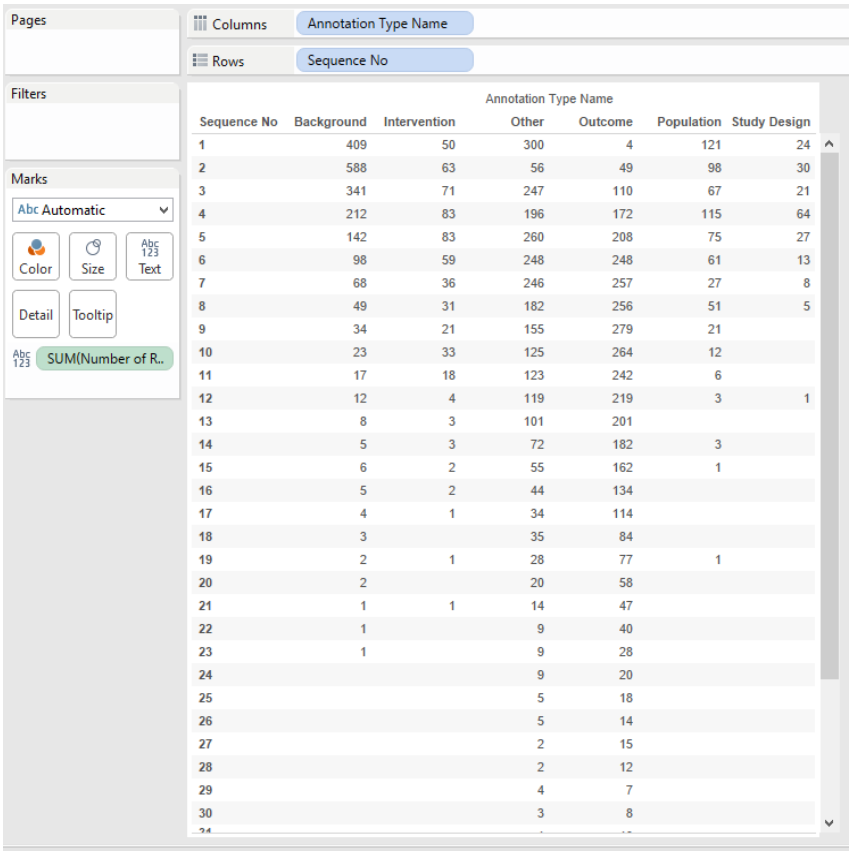


Figure 7.3: Stack Bar – PIBOSO annotations in structured and unstructured abstracts

Figure 7.4 shows the number of annotations per sentence sequence number for each annotation type while Figure 7.5 shows the same data in a stacked chart. Both Figure 7.4 and Figure 7.5 show that Population and Intervention annotations are almost always found in the first eleven sentences in abstracts. We can therefore define elementary annotation rules to reject Population and Intervention annotation candidates identified after the 11th sentence. This rule will sacrifice positive annotations that exist after the 11th sentence but this should have a minor impact as

the percentage of those annotations is very low compared to the ones found in the first 11 sentences.



	Annotation Type Name					
Sequence No	Background	Intervention	Other	Outcome	Population	Study Design
1	409	50	300	4	121	24
2	588	63	56	49	98	30
3	341	71	247	110	67	21
4	212	83	196	172	115	64
5	142	83	260	208	75	27
6	98	59	248	248	61	13
7	68	36	246	257	27	8
8	49	31	182	256	51	5
9	34	21	155	279	21	
10	23	33	125	264	12	
11	17	18	123	242	6	
12	12	4	119	219	3	1
13	8	3	101	201		
14	5	3	72	182	3	
15	6	2	55	162	1	
16	5	2	44	134		
17	4	1	34	114		
18	3		35	84		
19	2	1	28	77	1	
20	2		20	58		
21	1	1	14	47		
22	1		9	40		
23	1		9	28		
24			9	20		
25			5	18		
26			5	14		
27			2	15		
28			2	12		
29			4	7		
30			3	8		

Figure 7.4: PIBOSO sentences breakdown by their position within abstracts

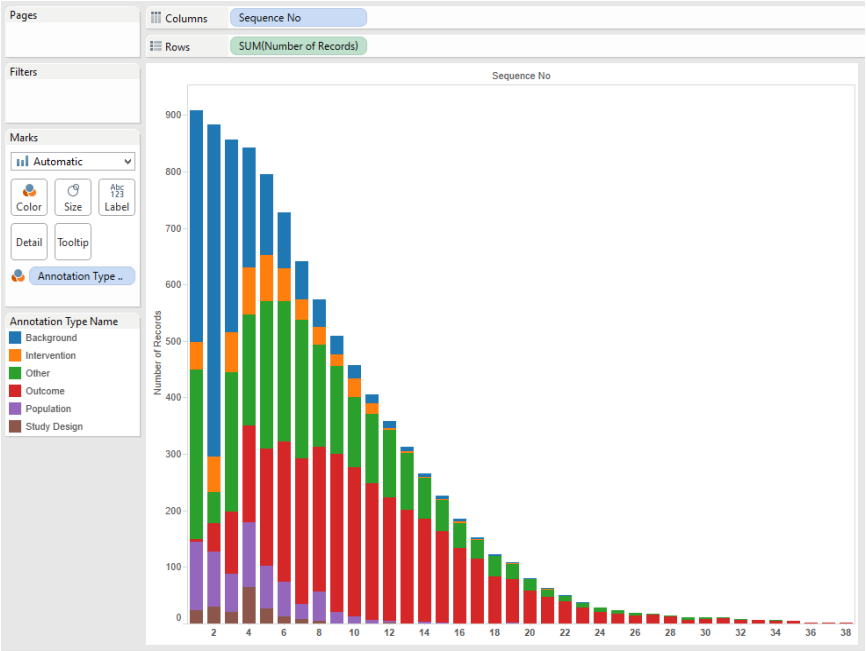
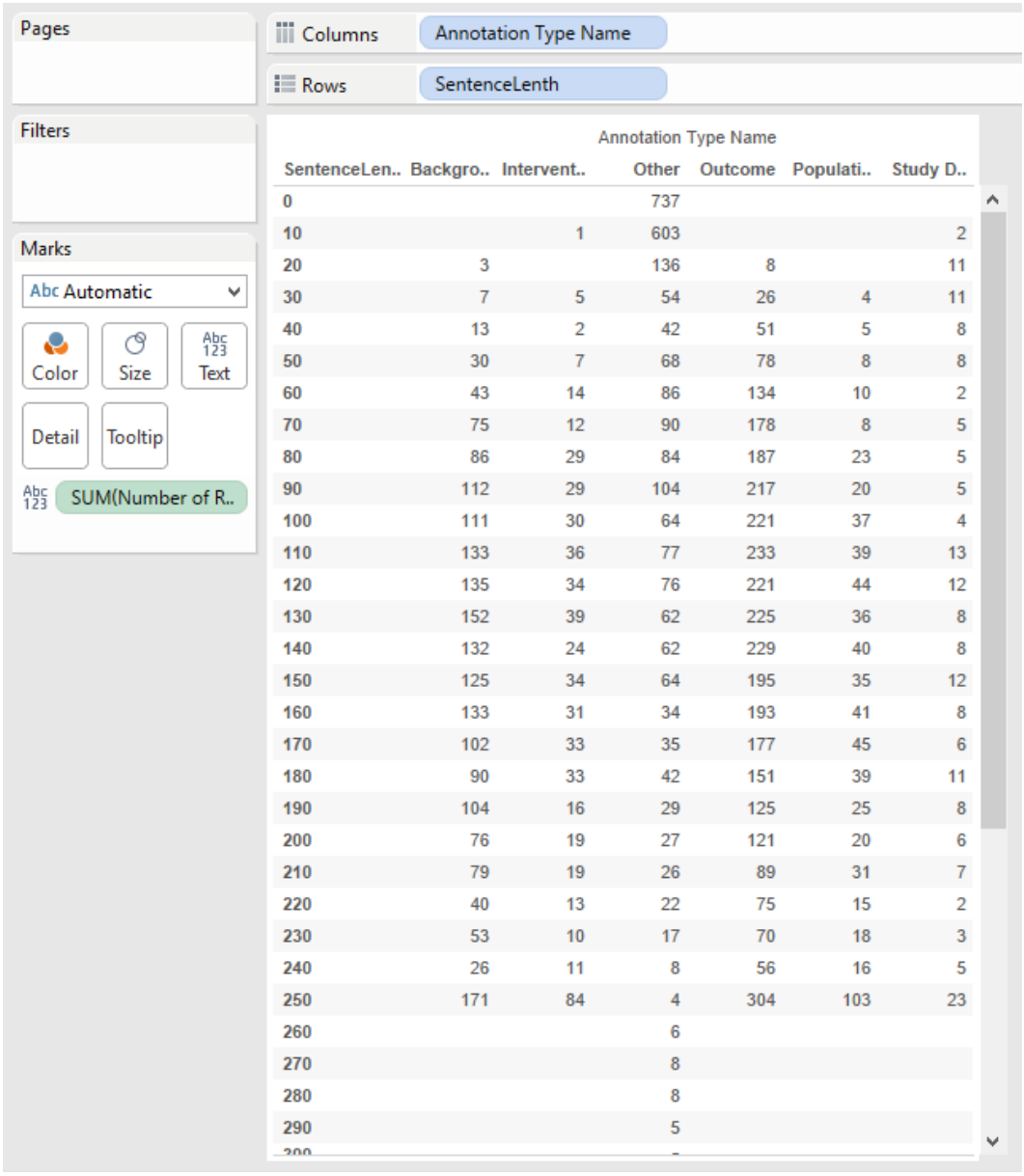


Figure 7.5: Stacked bar – PIBOSO sentences breakdown by their position within abstracts

Another interesting breakdown is the distribution of the different type of PIBOSO sentences based on their length. A sentence length is the number of characters in the sentence including white spaces and special characters. Since the length can be too granular for visualization, sentences were grouped into ranges of 0 to 10 characters for visualization. Figure 7.6 shows the PIBOSO sentence distribution based on their length range. Figure 7.7 shows the same data in a stacked bar chart. A number of observations can be made by looking at Figure 7.6 and Figure 7.7. An obvious observation is that short sentences (length <= 20) are almost always of type “Other”. Likewise, all sentences with more than 260 characters are of type “Other”. This information can be translated into an elementary annotation rule where a sentence is automatically labeled with the “Other” annotation type if its length is less than or equal to 20 or when its length is over 260. This rule alone can lead to an F-Score of 65 or more for the “Other” annotation type.



Annotation Type Name

SentenceLen..	Backgro..	Intervent..	Other	Outcome	Populati..	Study D..
0			737			
10		1	603			2
20	3		136	8		11
30	7	5	54	26	4	11
40	13	2	42	51	5	8
50	30	7	68	78	8	8
60	43	14	86	134	10	2
70	75	12	90	178	8	5
80	86	29	84	187	23	5
90	112	29	104	217	20	5
100	111	30	64	221	37	4
110	133	36	77	233	39	13
120	135	34	76	221	44	12
130	152	39	62	225	36	8
140	132	24	62	229	40	8
150	125	34	64	195	35	12
160	133	31	34	193	41	8
170	102	33	35	177	45	6
180	90	33	42	151	39	11
190	104	16	29	125	25	8
200	76	19	27	121	20	6
210	79	19	26	89	31	7
220	40	13	22	75	15	2
230	53	10	17	70	18	3
240	26	11	8	56	16	5
250	171	84	4	304	103	23
260			6			
270			8			
280			8			
290			5			

Figure 7.6: PIBOSO sentences breakdown by their length

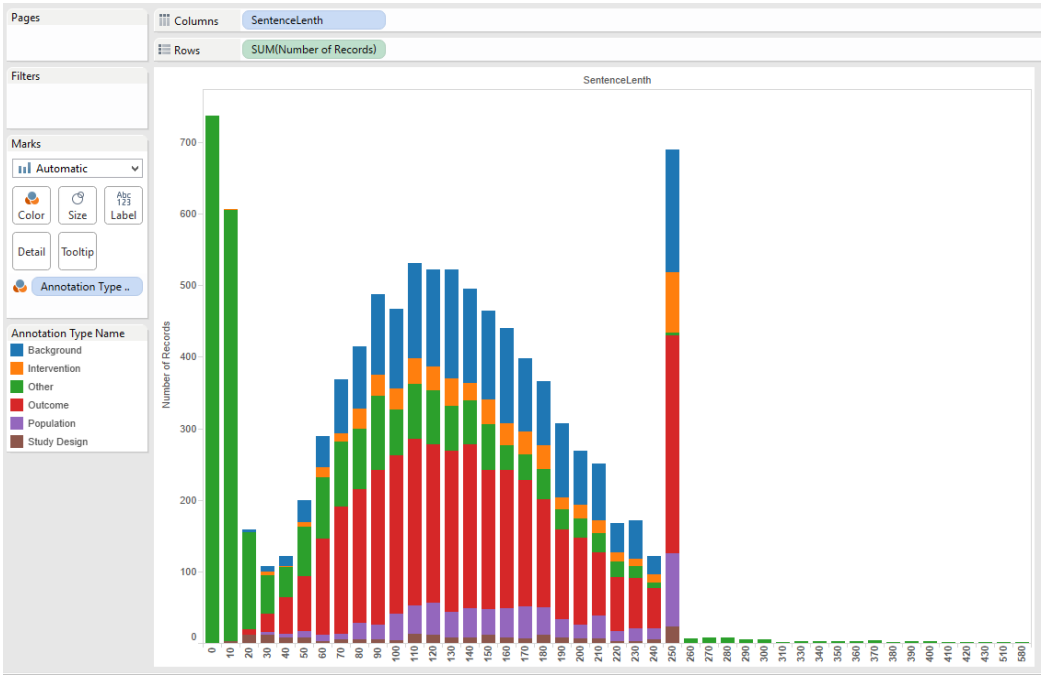
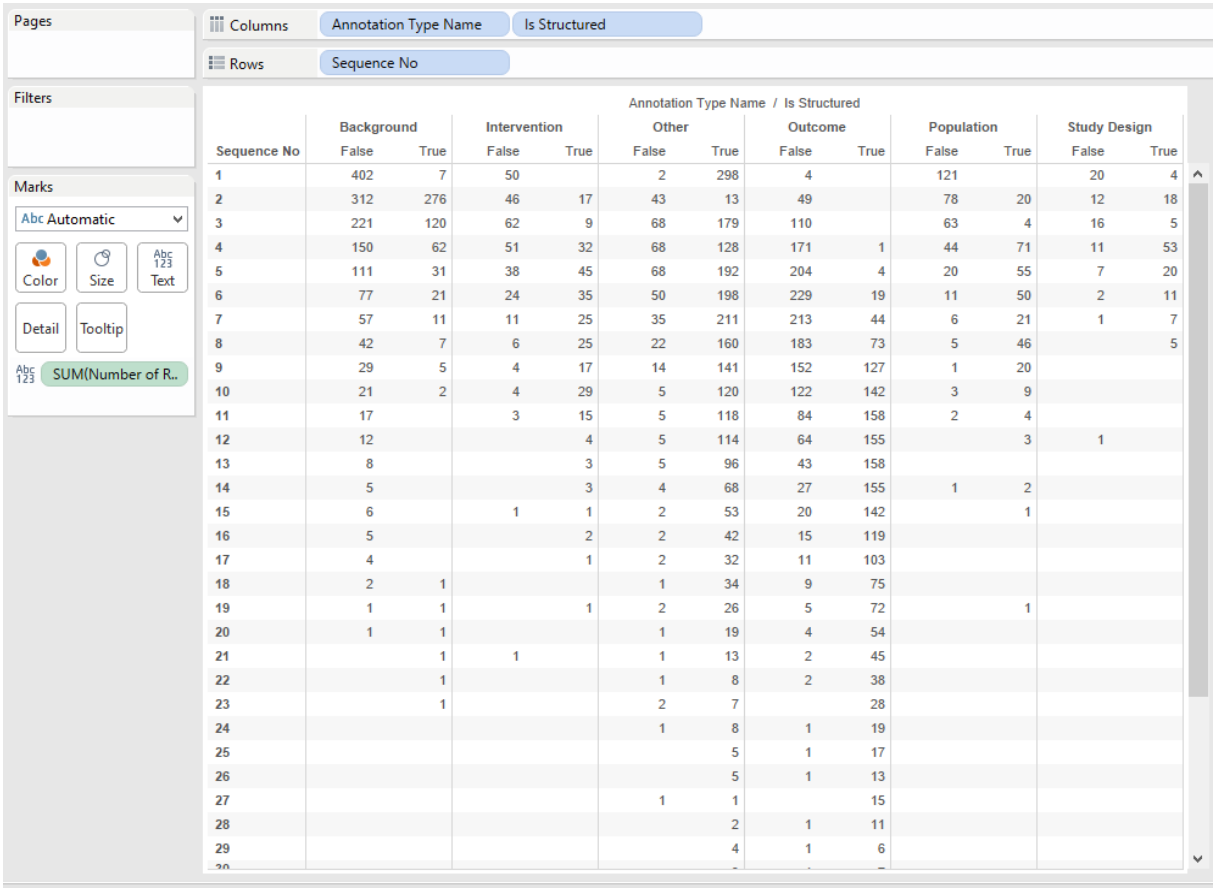


Figure 7.7: Stacked bar – PIBOSO sentences breakdown by their length

Adding more dimensions to the data analysis and visualization can sometimes reveal more details about the data and more importantly about the distinguishing factors between the various annotation types. For example, sentences positions and lengths in structured abstracts may not be quite the same as in unstructured abstract. In that case, it might be better to develop two sets of annotation rules (one for structured and one for unstructured abstracts). New dimensions can be analysed and visualized in isolation or side by side. With the isolation approach, only structured or unstructured abstracts are loaded for analysis and visualization in order to identify common properties. When put side by side, properties of each group of abstracts can be compared and contrasted with those of the other group. This helps determining whether one set of rules applies to both or not.

The remaining Figures in this section show numbers and graphics for structured and unstructured abstracts side by side. Figure 7.8 shows the number of each type of PIBOSO sentences at each position within structured and unstructured abstracts. Figure 7.9 is a visual presentation of the data in Figure 7.8 using a stacked bar chart. Both figures reveal important facts about the distribution of Population and Intervention annotations in structured and

unstructured abstract respectively. The biggest majority of Population and Intervention annotations in unstructured abstracts are in the first 6 sentences unlike the structured abstracts where the majority of annotations are in the second sentence and in sentences 4 to 10. This information helps writing better annotation rules to increase the number of true positives while reducing the number of false positives for Population and Intervention annotations.



Annotation Type Name / Is Structured

Sequence No	Background		Intervention		Other		Outcome		Population		Study Design	
	False	True	False	True	False	True	False	True	False	True	False	True
1	402	7	50		2	298	4		121		20	4
2	312	276	46	17	43	13	49		78	20	12	18
3	221	120	62	9	68	179	110		63	4	16	5
4	150	62	51	32	68	128	171	1	44	71	11	53
5	111	31	38	45	68	192	204	4	20	55	7	20
6	77	21	24	35	50	198	229	19	11	50	2	11
7	57	11	11	25	35	211	213	44	6	21	1	7
8	42	7	6	25	22	160	183	73	5	46		5
9	29	5	4	17	14	141	152	127	1	20		
10	21	2	4	29	5	120	122	142	3	9		
11	17		3	15	5	118	84	158	2	4		
12	12			4	5	114	64	155		3	1	
13	8			3	5	96	43	158				
14	5			3	4	68	27	155	1	2		
15	6		1	1	2	53	20	142		1		
16	5			2	2	42	15	119				
17	4			1	2	32	11	103				
18	2	1			1	34	9	75				
19	1	1		1	2	26	5	72		1		
20	1	1			1	19	4	54				
21		1	1		1	13	2	45				
22		1			1	8	2	38				
23		1			2	7		28				
24					1	8	1	19				
25						5	1	17				
26						5	1	13				
27					1	1		15				
28						2	1	11				
29						4	1	6				

Figure 7.8: PIBOSO sentences breakdown by their position in structured/unstructured abstracts

More useful observations about other annotation types can also be made by studying Figure 7.8 and Figure 7.9. For example, it is easy to notice that “Outcome” annotations are almost always in the first 14 sentences in unstructured abstract but they are mostly in sentences 7 to 22 in the structured ones. Similar observations can be made for other annotation types.

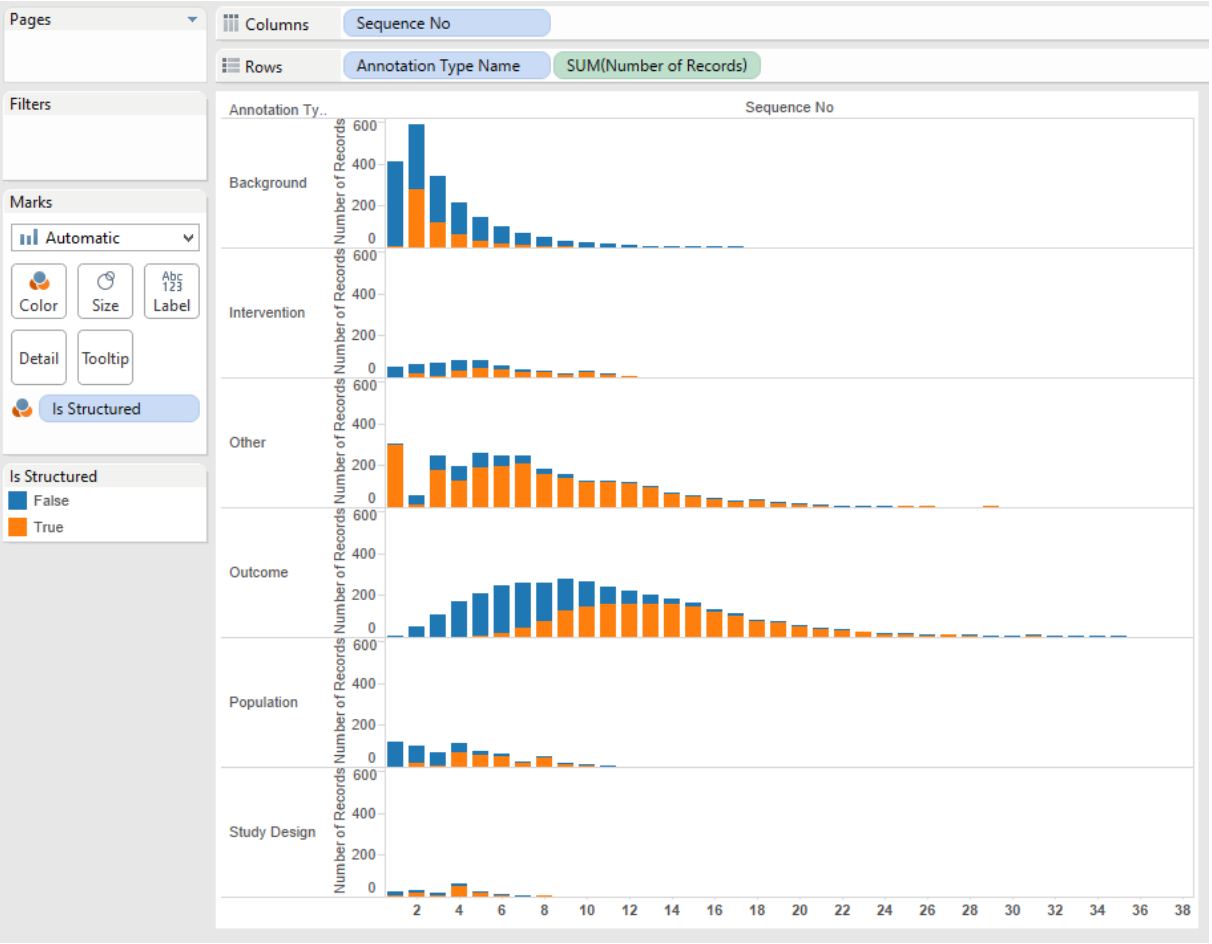
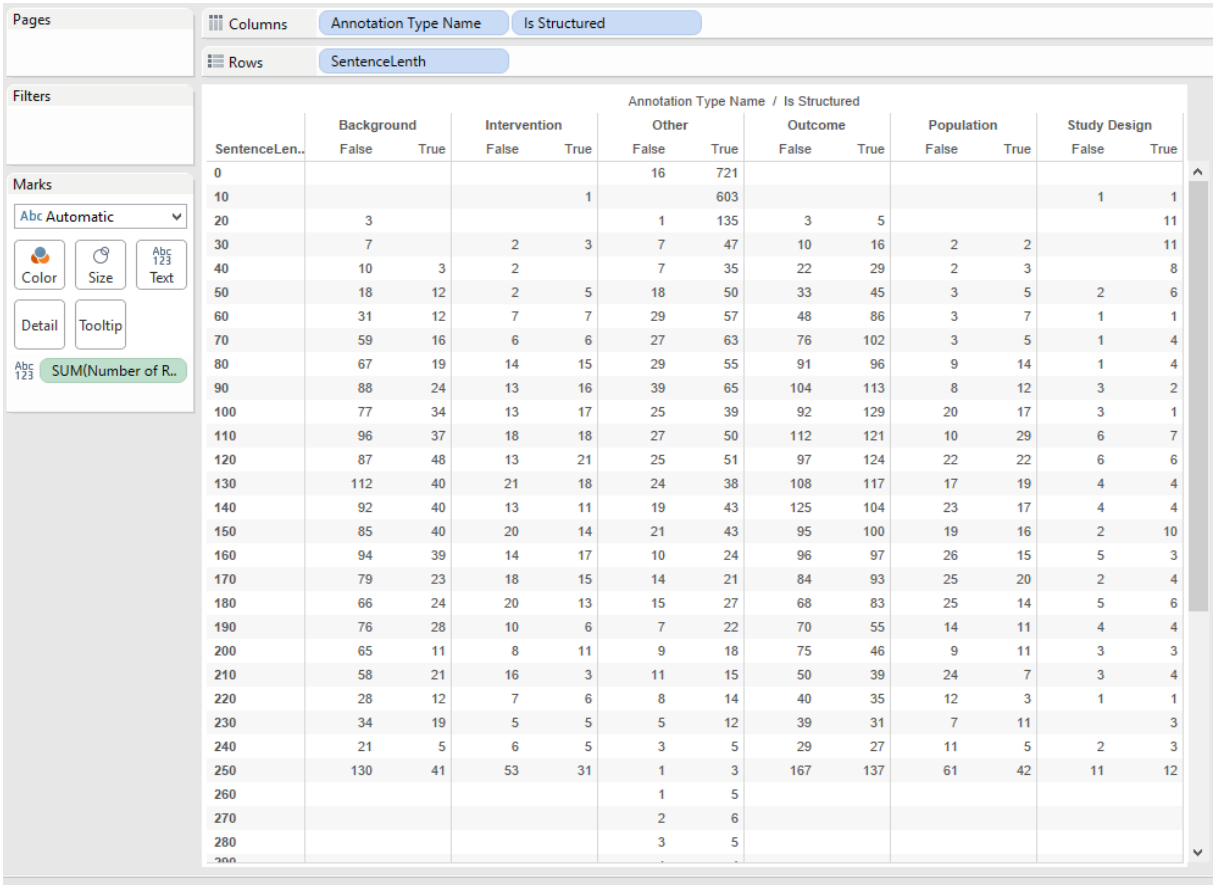


Figure 7.9: Stacked bar – PIBOSO sentences breakdown by their position in structured/unstructured abstracts

Figure 7.10 shows the PIBOSO annotations, in the training set for structured and unstructured abstracts, grouped by the length ranges of their sentences. Figure 7.11 shows the same data using a stacked bar graph. Both figures show that sentence lengths for Population and Intervention annotations in structured abstracts are consistent with those in the unstructured ones. This means that one rule set about the sentence lengths for each annotation type can be applied to both structured and unstructured abstracts. This is clearly not the case for annotations of type “Other” where the biggest majority of annotation sentences are less than 20 characters long in structured abstract as opposed to those in unstructured abstracts where lengths ranges between 50 and 180 characters.



Annotation Type Name / Is Structured

SentenceLen..	Background		Intervention		Other		Outcome		Population		Study Design	
	False	True	False	True	False	True	False	True	False	True	False	True
0					16	721						
10				1		603					1	1
20	3				1	135	3	5				11
30	7		2	3	7	47	10	16	2	2		11
40	10	3	2		7	35	22	29	2	3		8
50	18	12	2	5	18	50	33	45	3	5	2	6
60	31	12	7	7	29	57	48	86	3	7	1	1
70	59	16	6	6	27	63	76	102	3	5	1	4
80	67	19	14	15	29	55	91	96	9	14	1	4
90	88	24	13	16	39	65	104	113	8	12	3	2
100	77	34	13	17	25	39	92	129	20	17	3	1
110	96	37	18	18	27	50	112	121	10	29	6	7
120	87	48	13	21	25	51	97	124	22	22	6	6
130	112	40	21	18	24	38	108	117	17	19	4	4
140	92	40	13	11	19	43	125	104	23	17	4	4
150	85	40	20	14	21	43	95	100	19	16	2	10
160	94	39	14	17	10	24	96	97	26	15	5	3
170	79	23	18	15	14	21	84	93	25	20	2	4
180	66	24	20	13	15	27	68	83	25	14	5	6
190	76	28	10	6	7	22	70	55	14	11	4	4
200	65	11	8	11	9	18	75	46	9	11	3	3
210	58	21	16	3	11	15	50	39	24	7	3	4
220	28	12	7	6	8	14	40	35	12	3	1	1
230	34	19	5	5	5	12	39	31	7	11		3
240	21	5	6	5	3	5	29	27	11	5	2	3
250	130	41	53	31	1	3	167	137	61	42	11	12
260					1	5						
270					2	6						
280					3	5						

Figure 7.10: PIBOSO annotations breakdown by their sentence length in structured/unstructured abstracts

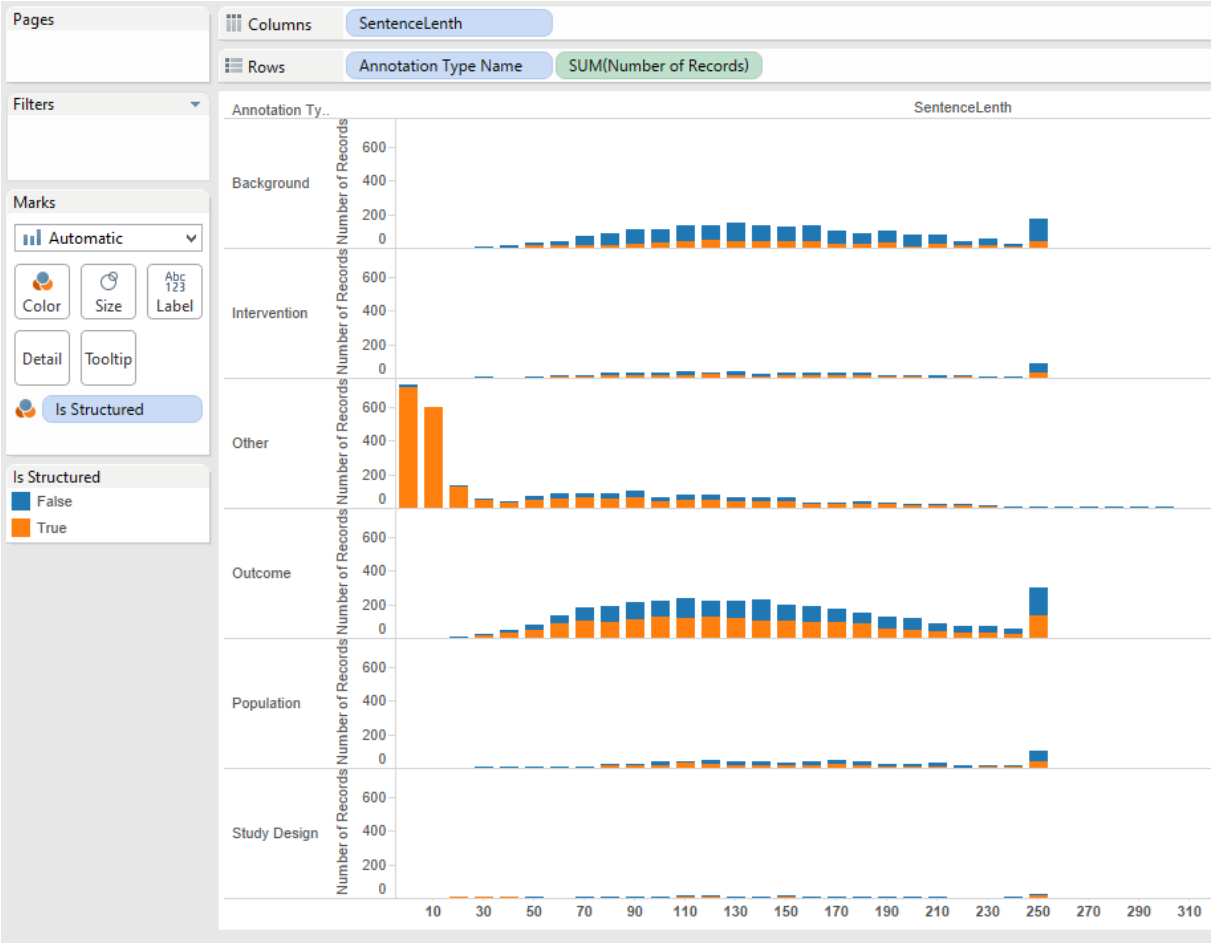


Figure 7.11: Stacked bar - PIBOSO sentences breakdown by their length in structured and unstructured abstracts

More analysis can be done in Tableau, in similar tools, or at the database level to retrieve more facts about the training data in order to build the initial set of annotation rules. One can, for example, see the number of overlapping annotations vs distinct ones. Simple SQL queries like the one in

Table 7.3 we ran over the PIBOSO training set in the ARDAKE database yields the following results:

- 216 common Population and Intervention sentences
- 178 Population sentences that are not Intervention sentences
- 129 Intervention sentences that are not Population sentences

Table 7.3: SQL snippet for common Population and Intervention annotations

```
-- Manual Test Run
DECLARE @TestRunID UniqueIdentifier = '88EF3C87-68C7-E311-BEAA-6036DDE93D1E'
DECLARE @AnnotationTypeID1 INT = 1 -- Population
DECLARE @AnnotationTypeID2 INT = 2 -- Intervention
DECLARE @DocumentTypeID INT = 1 -- Training

SELECT COUNT(*) [Count]
FROM [dbo].[XRef_Sentences_Annotations] ann1
INNER JOIN [dbo].[XRef_Sentences_Annotations] ann2
       ON ann1.SentenceID = ann2.SentenceID
INNER JOIN dbo.Sentences ps ON ps.SentenceID = ann1.SentenceID
INNER JOIN dbo.Documents d ON d.DocumentID = ps.DocumentID
WHERE
       (@DocumentTypeID IS NULL OR (@DocumentTypeID = d.DocumentTypeID))
       AND ann1.TestRunID = ann2.TestRunID
       AND ann1.TestRunID = @TestRunID
       AND ann1.AnnotationTypeID = @AnnotationTypeID1
```

This means that close to 55% of Population annotations are also Intervention annotations and about 63% of Intervention annotations are also Population annotations. This information can be taken into consideration while writing annotation rules so that when a Population annotation is identified with a high confidence, its confidence level for also being an Intervention annotation should be set to a minimum of 63%.

The above SQL query can be slightly modified to show common annotations grouped by position or by sentence length or any other available information. This helps writing more specific rules that lead to better results. The SQL code snippet in Table 7.4 lists common Population and Intervention annotations based on the position of each sentence within its parent abstract as shown in Table 7.5.

Table 7.4: SQL snippet for common Population and Intervention annotations by sentence position

```
-- Manual Test Run
DECLARE @TestRunID UniqueIdentifier = '88EF3C87-68C7-E311-BEAA-6036DDE93D1E'
DECLARE @AnnotationTypeID1 INT = 1 -- Population
DECLARE @AnnotationTypeID2 INT = 2 -- Intervention
DECLARE @DocumentTypeID INT = 1 -- Training
```

```
SELECT ps.SequenceNo [Sentence Position], COUNT(*) [Count]
FROM [dbo].[XRef_Sentences_Annotations] ann1
INNER JOIN [dbo].[XRef_Sentences_Annotations] ann2
ON ann1.SentenceID = ann2.SentenceID
INNER JOIN dbo.Sentences ps ON ps.SentenceID = ann1.SentenceID
INNER JOIN dbo.Documents d ON d.DocumentID = ps.DocumentID
WHERE (@DocumentTypeID IS NULL OR (@DocumentTypeID = d.DocumentTypeID))
AND ann1.AnnotationTypeID != ann2.AnnotationTypeID
AND ann1.TestRunID = ann2.TestRunID AND ann1.TestRunID = @TestRunID
AND ann1.AnnotationTypeID != @AnnotationTypeID1
AND ann2.AnnotationTypeID = @AnnotationTypeID2
GROUP BY SequenceNo
ORDER BY [Count] DESC
```

Table 7.5: Common Population and Intervention annotations by sentence position

Sentence Position	**Common Population and Intervention Count**
1	25
2	24
3	21
4	18
5	18
6	10
7	5
8	2
9	2
11	2
19	1
10	1

Table 7.5 shows that most common Population and Intervention annotations occur in the first 6 sentences. The above SQL query can be adjusted to eliminate rows with low counts.

Further breakdown can be done by including the structured/unstructured abstract to be even more specific about common Population/Intervention annotations.

7.5 Chapter Summary

Having some level of domain and corpus understanding is essential for any KE project. This is especially true for KEFUD projects where unsupervised ML algorithms are quite limited and where it is hard or impossible to create a training set.

In this chapter, we showed how existing visualization tools along with our analysis tools gave us a quick insight and helped us understanding the PIBOSO corpus. This was a big step forward towards defining our initial KE rules that we describe in the next chapter.

CHAPTER 8

Rules Development

8.1 Chapter Overview

Having a good understanding of the domain and corpus is essential for determining what KE rules are needed. However, this effort becomes useless if we don't have the tools and capacity to create and deploy the required rules. Having the right set of tools to create KE rules can have a huge impact on the time and cost of rules development.

The tools we created as part of our ARDAKE prototype were so helpful in creating simple and efficient rules to identify Population and Intervention sentences in the PIBOSO corpus. We show in this chapter how we created textual, statistical, and semantic rules, using ARDAKE, in a simple and consistent way.

8.2 Building the PIBOSO Elementary Rules in ARDAKE

Based on the visual and non-visual analysis we did, we created a number of elementary rules to identify Population and Intervention annotations in the PIBOSO corpus. For example, it is common to mention an age or an age range in population sentences. We therefore needed to build elementary annotation rules to match age and age range patterns. An age or age range pattern in a sentence gives an indication that this may be a population sentence. To increase the confidence level, the same sentence containing an age or an age range pattern is searched for other population related patterns. This is done by building more elementary annotation rules and combining these rules to calculate the final confidence level to decide whether or not the sentence is a population annotation.

Building annotation rules in ARDAKE is done by simple mouse drag/drop of predefined built-in or user defined patterns, conditions, and actions. Figure 8.1 (A) shows an ARDAKE rule for matching age patterns. The AgeUnit pattern in the rule matches any age unit such as year, month, day, etc. The AgeKeyword pattern is either "old" or "of age". The rule in Figure 8.1

(A) defines the age as a number (written in digits or in letters) followed by any characters (spaces or other) followed by an age unit, then any characters and ends with an age keyword. This allows matching age patterns in different forms such as “3 weeks old”, “fifty three years of age”, etc.

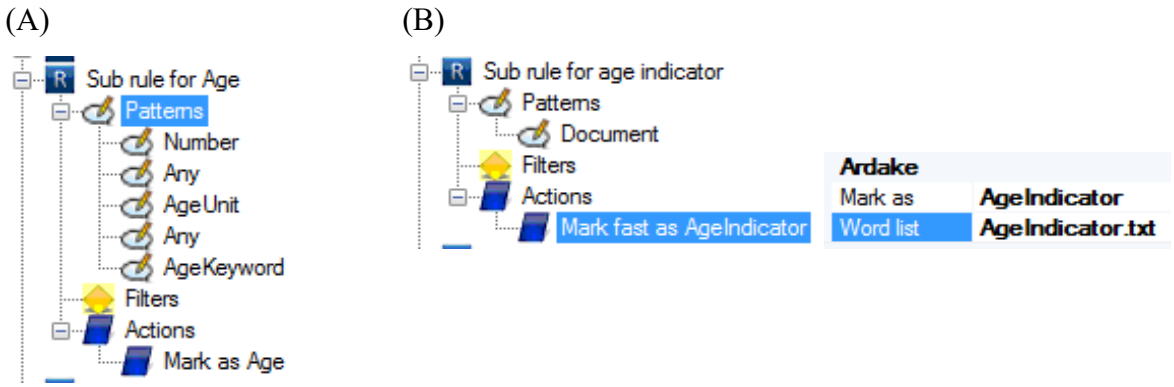


Figure 8.1: Age Rule (A) and Age Indicator Rule (B) in ARDAKE

Figure 8.1 (B) shows another ARDAKE rule to match age indicators in sentences. This rule uses the “Mark fast” action that matches any word from a selected word list. Our age indicator word list contains all variations of English age indicators such as “infant”, “toddler”, “baby”, “child”, “teenager”, etc…, in their singular and plural forms.

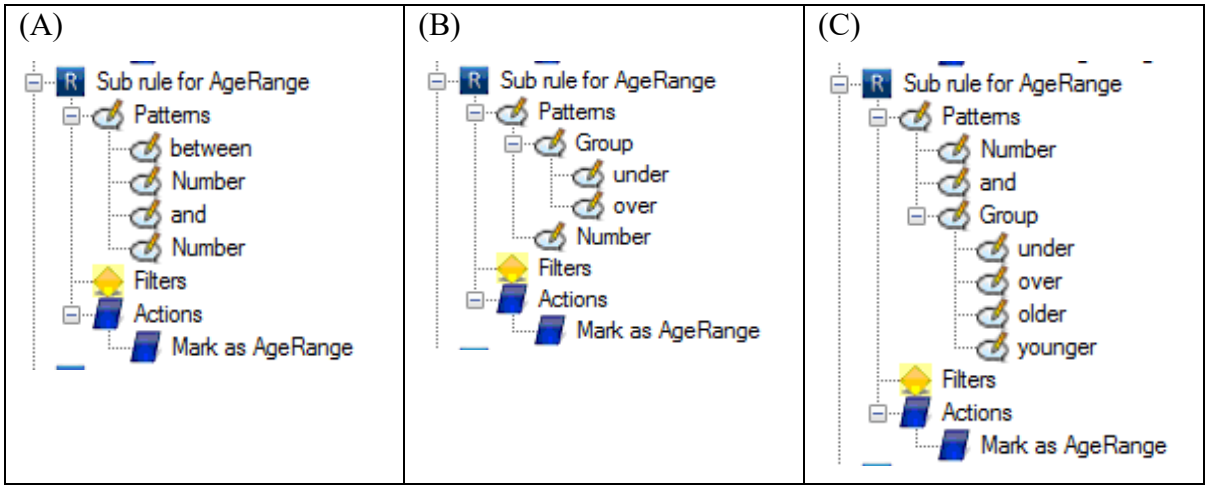


Figure 8.2: Age Range Rules in ARDAKE

The rules in Figure 8.2 match age ranges in different forms including “between 30 and 40”, “between ten and eleven”, “under twenty”, “16 and over”, and so on.

The ARDAKE “Mark fast” action can be used to match positive and negative n-grams. All you need is to save positive and negative n-grams, identified by the ARDAKE Corpus Analyser, in text files then set the “Word List” property of the “Mark fast” action to point to those files. Once this is done, these two rules can be used to define a new rule to identify sentences that have one or more positive population n-grams and no negative population n-grams as a population candidate sentence as shown in Figure 8.3.

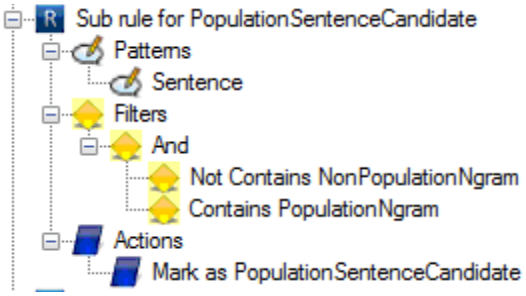


Figure 8.3: Population Sentence Candidate Rule in ARDAKE

PIBOSO Semantic rules are as easy to create as the previously shown ones in this section. It is common for population sentences to include the problem or the disease of interest. A generic ARDAKE annotation rule can be created to match any disease in the text being analysed. This can be done by simply using the “Subclass of” condition and setting the “Parent Concept” property to the top-most disease class in MeSH or a similar ontology. However, this can generate too many concepts to look for and can result in matching many false positive results. To obtain better results, we should be as specific as possible about the concepts (diseases or problems) to match in the text. Ideally, a dedicated ontology for the corpus being analysed should be used but, since we could not find such an Ontology for the PIBOSO corpus and it would be a tedious task to develop one, we decided to use the MeSH ontology and be more specific about the parent concepts to look for in population sentences. The main reason for

choosing MeSH over other medical ontologies is that NLM uses the MeSH to index the articles in the MedLine/PubMED database.

By reading some of the PIBOSO training population annotations, you can notice that they are mostly about spinal cord issues and brain injuries. Figure 8.4 shows a rule that defines any occurrence of a sub-concept of the “Spinal Cord Diseases” or “Brain Injuries” as a PIBOSO disease.

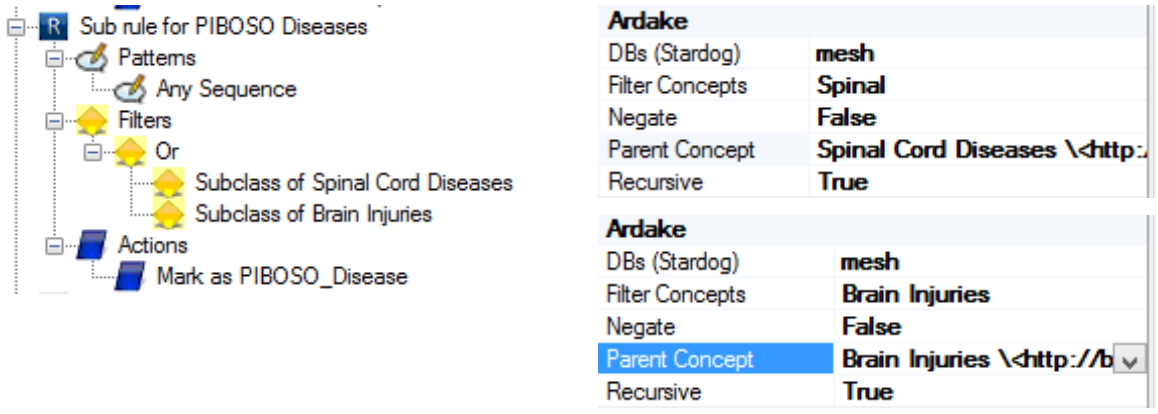


Figure 8.4: PIBOSO Diseases Annotation Rule in ARDAKE

Other population related rules such as the ones identified during the visual analysis are trivial to create in ARDAKE. This is done by using the Position or Length conditions as shown in Figure 8.5.

The rule in Figure 8.5 covers frequent positions of population sentences in both structured and unstructured abstracts. For better results, the rule in Figure 8.5 could be replaced with two other rules, one for structured abstracts that looks for sentences at positions (2, 4, 5, 6, 7, 8, 9, and 10) and one for unstructured abstracts that looks for sentences at positions 1 to 6 inclusively. This is based on numbers and observations from Figure 7.8 and Figure 7.9 in the previous chapter.

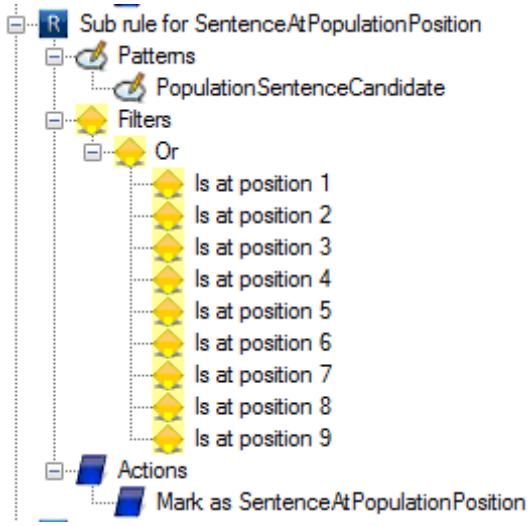


Figure 8.5: ARDAKE Rule to Identify Sentences at Population Position

Once all elementary rules are created, new rules can be created to assign scores to the results of each elementary rule. This is done using the Mark Score action as in Figure 8.6.

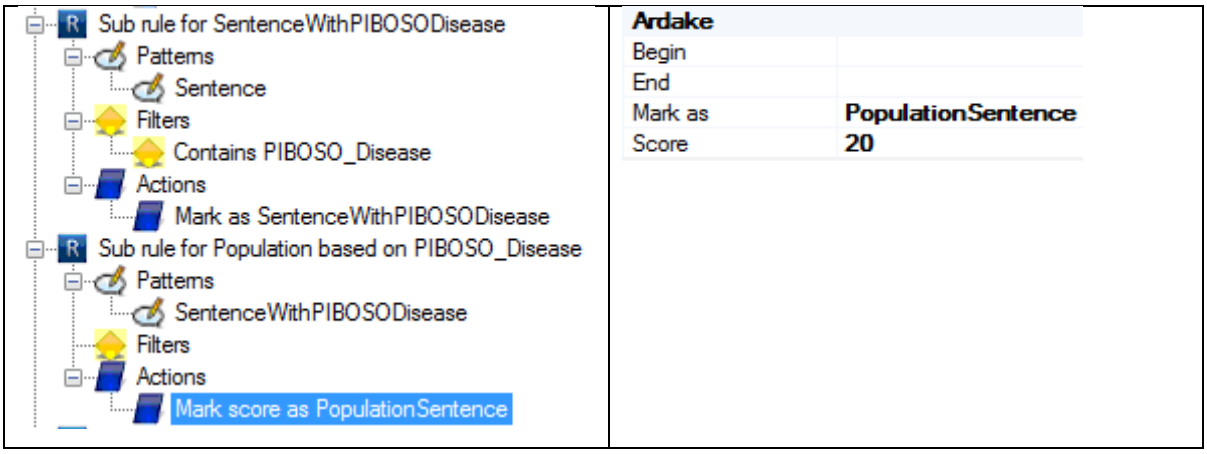


Figure 8.6: Mark Score Action for Sentences Containing PIBOSO Disease Annotations

Figure 8.6 shows two sub-rules where the first one is to annotate any sentence that contains a PIBOSO Disease pattern as SentenceWithPIBOSODisease and the second one to create an annotation of type PopulationSentence with a score of 20 for each SentenceWithPIBOSODisease. This is because the second sub-rule does not have any conditions. Note that if a sentence was already annotated as a PopulationSentence by another

sub-rule and that this same sentence also contains a PIBOSO_Disease annotation, the score of the PopulationSentence annotation is increased by 20.

8.3 Running the Elementary Rules

ARDAKE allows running rules by clicking the Run button or generate UIMA Ruta scripts from its visual rules. This is done by simply choosing the “Generate Ruta Script” button from the ARDAKE menu bar. Generated Ruta scripts like the snippet in Table 8.1 can then be run into any UIMA environment.

Table 8.1: UIMA Ruta script snippet generated by ARDAKE

```
DECLARE AgeIndicator;
WORDLIST AgeIndicator = 'AgeIndicator.txt';
Document{->MARKFAST(AgeIndicator, AgeIndicator)};

DECLARE AgeKeyword;
WORDLIST AgeKeyword = 'AgeKeyword.txt';
Document{->MARKFAST(AgeKeyword, AgeKeyword)};

DECLARE AgeUnit;
WORDLIST AgeUnit = 'AgeUnit.txt';
Document{->MARKFAST(AgeUnit, AgeUnit)};

DECLARE Gender;
WORDLIST GenderList = 'GenderList.txt';
Document{->MARKFAST(Gender, GenderList)};

DECLARE Age;
Number ANY? AgeUnit ANY? AgeKeyword{->MARK(Age, 1, 5)};

DECLARE AgeRange;
"between" Number "and" Number{->MARK(AgeRange, 1, 4)};

("under" | "over") Number{->MARK(AgeRange, 1, 2);

Number "and" ("under" | "over" | "older" | "younger"){->MARK(AgeRange, 1, 3)};

DECLARE PopulationNgram;
```

```
WORDLIST P_Indicators = 'P_Indicators.txt';
Document{->MARKFAST(PopulationNgram, P_Indicators)};

DECLARE NonPopulationNgram;
WORDLIST P_Negators = 'P_Negators.txt';
Document{->MARKFAST(NonPopulationNgram, P_Negators)};

DECLARE PopulationSentenceCandidate;
Sentence{AND(-CONTAINS(NonPopulationNgram), CONTAINS(PopulationNgram))
->MARK(PopulationSentenceCandidate)};

DECLARE SentenceAtPopulationPosition;
PopulationSentenceCandidate{
OR(
POSITION(Document, 1), POSITION(Document, 2), POSITION(Document, 3),
POSITION(Document, 4), POSITION(Document, 5), POSITION(Document, 6),
POSITION(Document, 7), POSITION(Document, 8), POSITION(Document, 9)
) ->MARK(SentenceAtPopulationPosition)};

DECLARE PIBOSO_Disease;
#{
OR(
SubClassOf("<http://bioonto.de/mesh.owl#C10.228.854>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#C10.228.140.199>", "mesh", true)
) ->MARK(PIBOSO_Disease)};
```

Different options exist when it comes to running elementary rules and getting their results. We could generate a separate Ruta script for each elementary rule then run this script and import its resulting annotations into the ARDAKE database for further analysis and improvements. This option is best suitable for business users as it does not require any knowledge about Ruta or the database. Another option is to generate one script with all elementary rules then comment out all elementary rules except one in the Ruta script before running and importing the annotations. Once the results are imported, the active elementary rule is commented out and another one uncommented then the script is run again until all rules are executed and their results are imported in the database. Unlike the first option, the second option requires some

basic Ruta knowledge. A third option is to generate one Ruta script with all the elementary annotation rules, run this script once, then do the analysis on the annotation types produced by each elementary rule instead of doing it based on the target annotation type (ex. Population or Intervention). The last option takes less time than the first two but requires some basic database and SQL knowledge.

8.4 Chapter Summary

We showed the main rules we created using ARDAKE to identify Population and Intervention sentences in the PIBOSO corpus. Although these rules were of different types including textual, statistical, and semantic rules, they were all created in a consistent and simple visual way. These are the same rules we used to obtain better KE results than those obtained by state-of-the-art KE algorithms when applied on the NICTA-PIBOSO corpus as detailed in the next chapter.

CHAPTER 9

Evaluation

9.1 Chapter Overview

Newly created KE rules and models should be tested, evaluated, and optimized to produce the best results. Sometimes it is even necessary to update rules that are already deployed especially when the business logic or the data change or if more information becomes available. To optimize rules, we first need to identify their shortcomings.

Creating and optimizing rules without having the right tools is difficult but finding the right rules combinations that yield the best KE results can be much more challenging.

In this chapter, we show how our 'Results Analyser and Visualizer' tool can be used to indicate what rules should be optimized and how to optimize them. We also show how we used our 'Rules Combiner' tool to automatically generate the best rules combination that outperformed most state-of-the-art KE algorithms when run over the NICTA-PIBOSO corpus to identify population and intervention sentences.

9.2 Measuring the performance of rules

As discussed in CHAPTER 2, the performance of rules can be measured in different ways using existing supervised machine learning evaluation methods. The most common evaluation methods/formulas are F-Score, ROC, AUC, Accuracy, Sensitivity (Recall), Specificity, and Precision [11]. Storing rules results in a database along with the training and test data makes it easier to measure and compare the performance of KE rules using visual tools and/or SQL queries.

We developed a number of SQL functions to calculate the precision, recall, and F-Score for annotation results stored in the in ARDAKE database. Other SQL functions and stored procedures in the ARDAKE database return the TP, FP, TN, and FN sets for any given test

run. Analyzing and visualizing the output of the ARDAKE database functions and stored procedures help optimizing the rules used to generate annotation results.

The ARDAKE Rules Results Analyser described in Section 5.3.7 leverages the SQL Server capabilities by presenting functions and stored procedures' results in a visual user-friendly way.

9.3 Creating, Visualizing, Analyzing, and Optimizing the Elementary rules for Population and Intervention

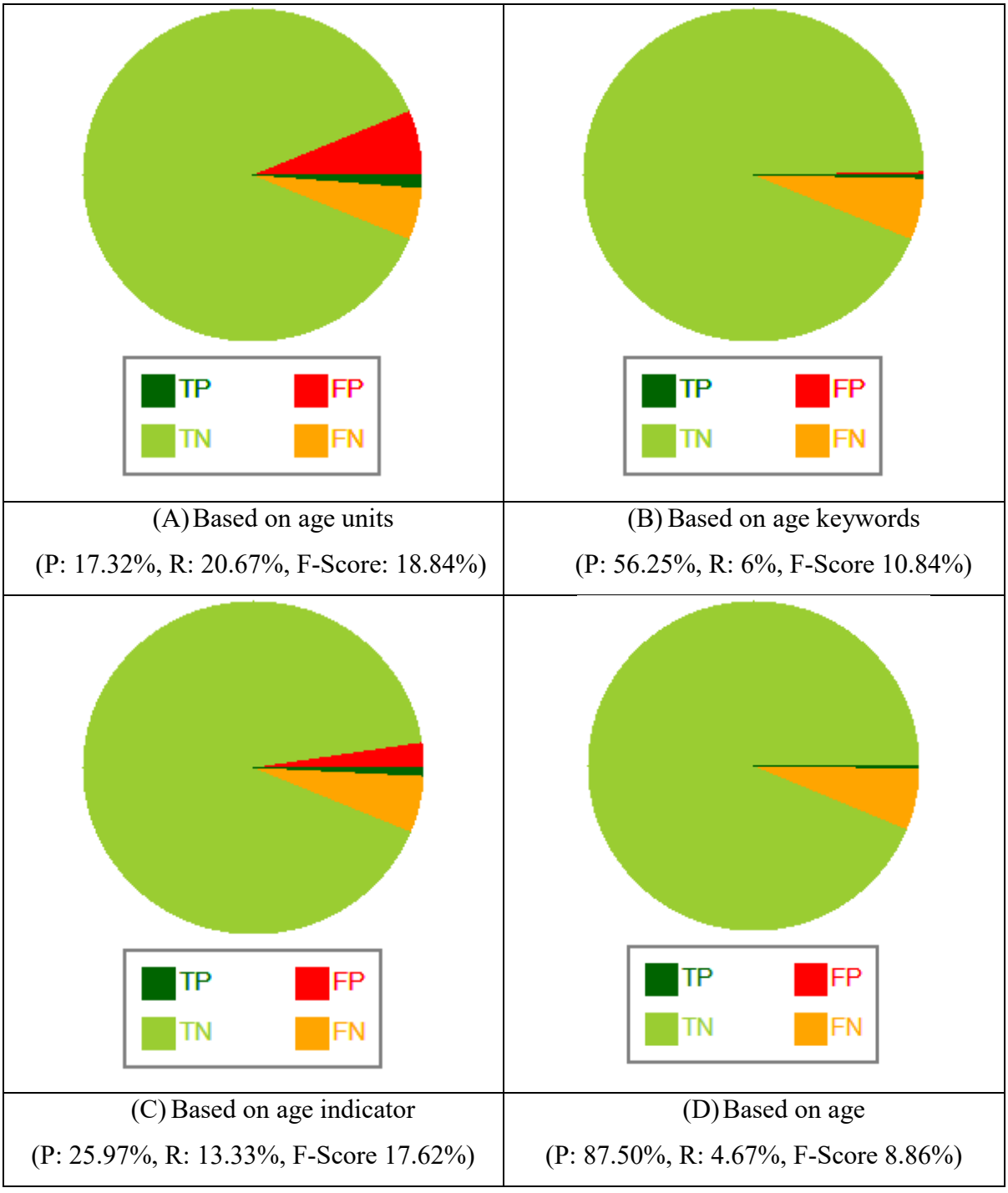
Population and Intervention rules for the PIBOSO corpus were iteratively developed starting with elementary rules to annotate simple patterns. Simple patterns definitions were obtained from two main sources:

- A research report, done at the University of Quebec in Outaouais, to define the characteristics of PICO terms.
- The results of analyzing PIBOSO training data as explained in CHAPTER 7.

The following subsections show the main Population elementary rules along with their precision, recall, and F-Score. Elementary Intervention rules were developed in the same manner.

9.3.1 Population annotation based on business rules

A research report prepared at the University of Quebec in Outaouais described the properties of patterns found in different PIBOSO sentences. For example, it is logical for sentences describing patients or population to have age and/or gender related patterns. We showed how these rules can be created using ARDAKE in CHAPTER 8. Figure 9.1 shows the visual representation of the annotation results for a number of age related rules along with the precision, recall, and F-Score of each rule. An interesting observation is that the results (precision, recall, and F-Score) of each business rule when run over the training set were almost the same results of this rule when run over the test set.



(A) Based on age units
(P: 17.32%, R: 20.67%, F-Score: 18.84%)

(B) Based on age keywords
(P: 56.25%, R: 6%, F-Score 10.84%)

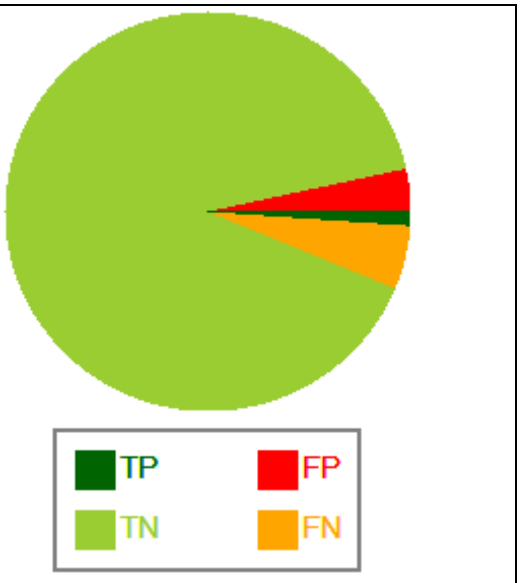
(C) Based on age indicator
(P: 25.97%, R: 13.33%, F-Score 17.62%)

(D) Based on age
(P: 87.50%, R: 4.67%, F-Score 8.86%)

Figure 9.1: Population annotation results based on age related rules

Note that the age units rule (Figure 9.1 (A)) has a poor precision and a good recall as opposed to the age keywords rule (Figure 9.1 (B)) that has a good precision and a poor recall. This indicates that age units are found in many Population sentences but are also present in many non-Population sentences. It also indicates that age keywords are found in few Population sentences but are very unlikely to show up in non-Population sentences. The age rule that combines both age units and age keywords has an excellent precision but a very low recall. Combining the age units and age keywords rules resulted in 1.33% recall loss but increase the precision by over 30% compared to the age keywords rule.

Some elementary rules such as the gender rule (Figure 9.2) lead to relatively acceptable precisions and good recalls.



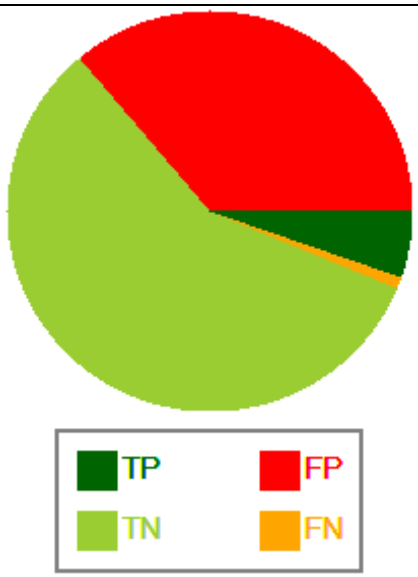
(P: 25.69%, R: 18.67%, F-Score 21.62%)

Figure 9.2: Population annotation results based on gender

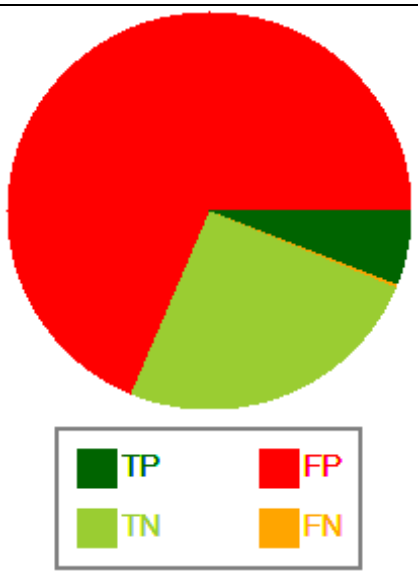
9.3.2 Population annotation based on statistical rules

Like business rules, statistical rules produced similar results whether they were run over the training or the test set of the NICTA-PIBOSO corpus. Figure 9.3 shows the visual

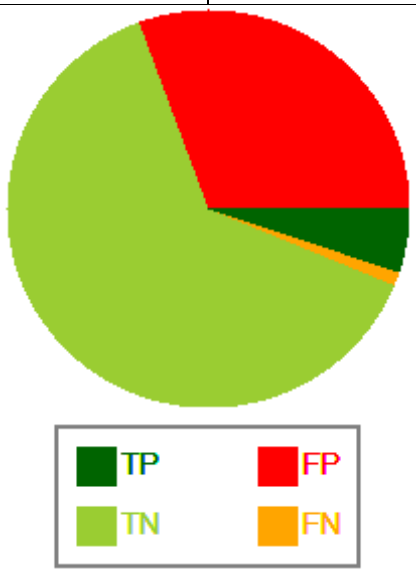
representation of results produced by the sentence position and length statistical rules and their combination using the AND operator.



A) Based on sentence position (1 to 9)
(P: 13%, R: 85.33%, F-Score 22.57%)



B) Based on length (80 to 260)
(P: 8.13%, R: 96%, F-Score 14.99%)



C) Based on sentence length and position
(P: 14.6%, R: 82.67%, F-Score 24.82%)

Figure 9.3 : Population annotation results based on statistical rules

Notice that the F-Score of the combination is higher than the F-Score of both elementary rules despite the lower recall. This is because the drop in the recall was compensated for by the precision increase. The results correspond to our analysis and observations in CHAPTER 7 where we show that almost all population sentences are amongst the first 9 sentences with a length ranging between 80 and 260 characters.

9.3.3 Population annotation based on inclusion and exclusion n-gram tree rules

The annotation results for rules based on n-gram decision trees, produced by the ARDAKE Corpus Analyser, revealed three interesting points:

1- The ARDAKE algorithm that generates n-gram decision trees is very powerful and useful for annotation and knowledge extraction especially when the training set properly represents the domain data.
2- The use of n-grams extracted from a training set negatively impacts the quality of KE models and rules if the training set does not properly represent the domain data.
3- The PIBOSO training set does not properly represent the corresponding test set as indicated in [51]. This is true, at least, from a linguistic perspective.

By looking at Figure 9.4 and Figure 9.5, we can easily see how using annotation rules with unigrams produced based on one set generates excellent results for the same set but poor results for the other set. Figure 9.6 shows that the annotation rule for unigrams produced based on both sets generated very good and identical results for both training and test sets. In all these cases, n-grams were generated based on precision but, as explained in CHAPTER 4, the ARDAKE Corpus Analyser can also generate n-grams based on the recall or the F-Score.

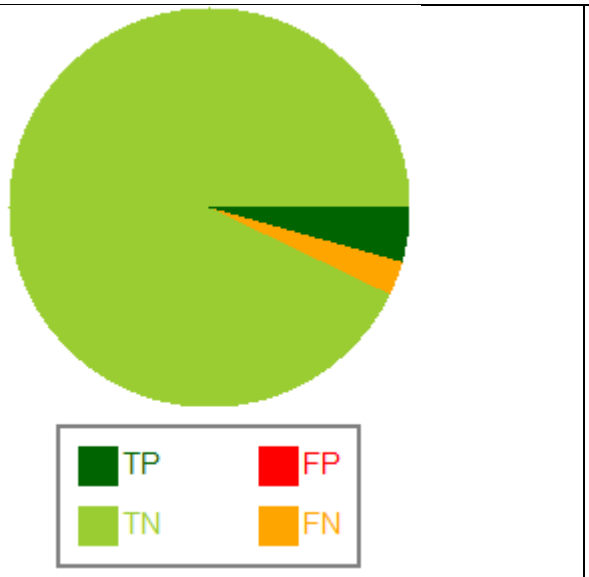
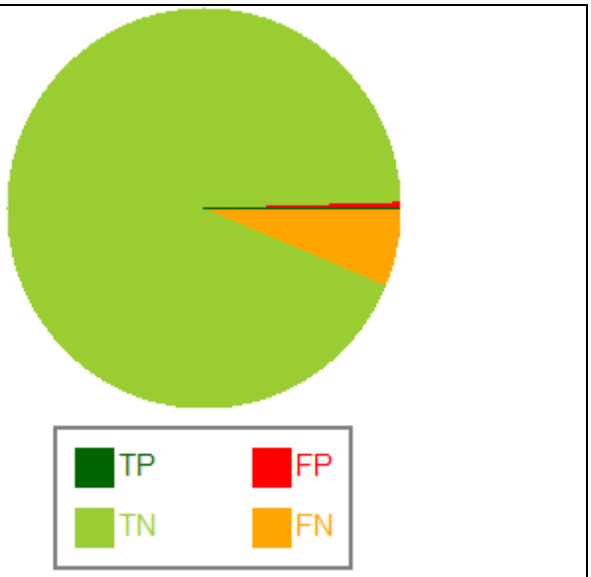
	
Training P: 99.75%, R: 61.48%, F-Score: 76.07%	Test P: 26.67%, R: 2.67%, F-Score: 4.85%

Figure 9.4: Results of a Population annotation rule based on unigrams from the Training set

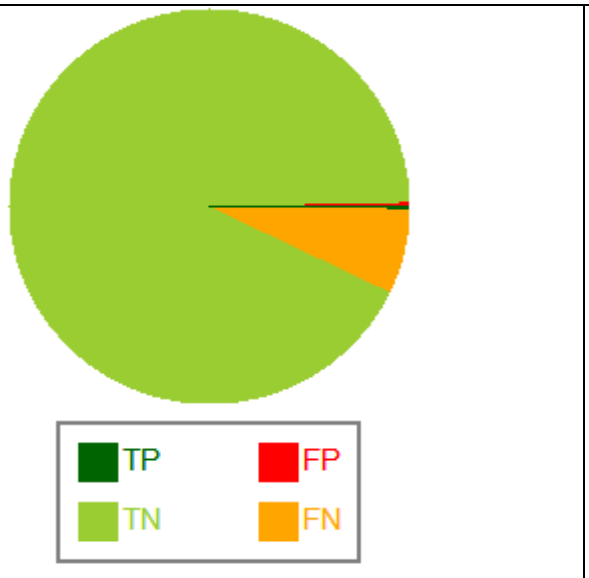
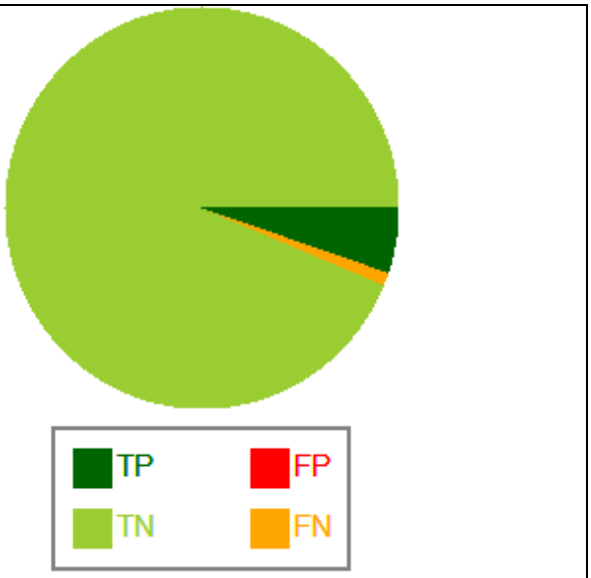
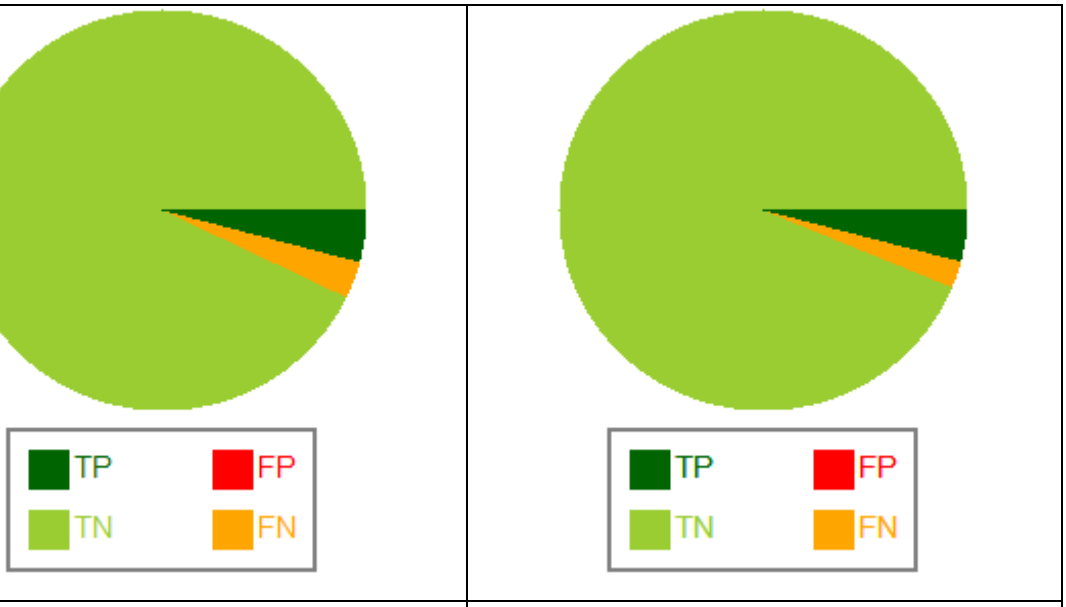
	
Training P: 35.42%, R: 2.57%, F-Score: 4.79%	Test P: 100.00%, R: 83.33%, F-Score: 90.91%

Figure 9.5: Results of a Population annotation rule based on unigrams from the Test sets

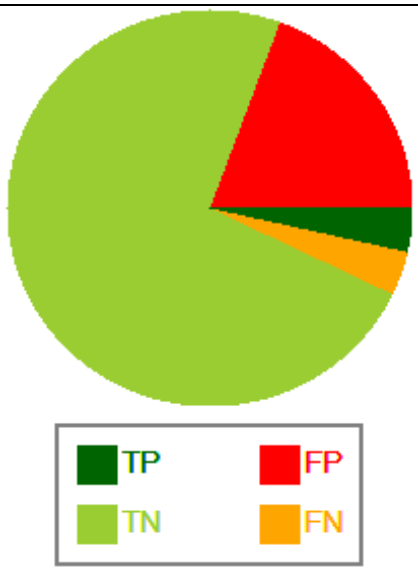


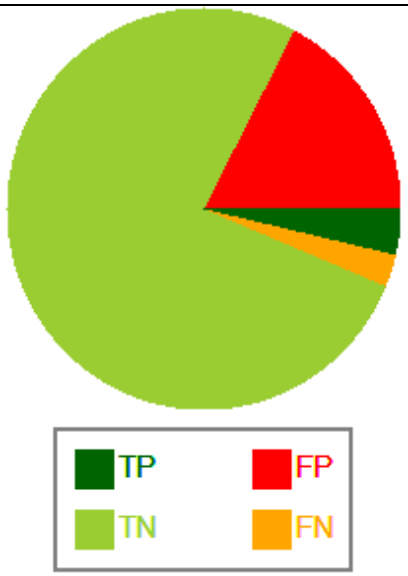
Training	Test
P: 99.74%, R: 58.46%, F-Score: 73.71%	P: 100.00%, R: 66.00%, F-Score: 79.52%

Figure 9.6: Results of a Population annotation rule based on unigrams from the Training and Test sets

9.3.4 Population annotation based on semantic rules

Like business and statistical rules, semantic rules produced similar results when run over the training and test sets of the PIBOSO corpus. This is normal because semantic rules can be considered as business rules that are defined based on ontologies. The graphs in Figure 9.7 show that more than half of the population sentences and less than the quarter of the non-population sentences contain a MeSH Disorder subclass.





Training	Test
P: 15.57%, R: 49.85%, F-Score: 23.72%	P: 17.60%, R: 58.67%, F-Score: 27.08%

Figure 9.7: Annotation results based on Disorder subclasses in the MeSH ontology.

9.4 Generating the best rules combination

Once all elementary rules are defined, tested individually, and had their results stored in the ARDAKE database, it is time to find the rules combination that yields the best performance based on the F-Score or other metrics.

Annotation rules can be combined using logical operators ‘AND’ and ‘OR’. The number of combinations increases exponentially with the number of initial rules. For example, given two rules A and B, there are only 2 combinations (A AND B) and (A OR B) which gives a total of 4 rules altogether. Adding only one rule C creates 6 new rule combinations that are (A AND B AND C), (A AND (B OR C)), (A OR (B AND C)), (A OR B OR C), ((A AND B) OR C), ((A OR B) AND C).

Testing rules combinations manually is very time consuming as it can take days to test few hundred combinations. The ARDAKE Rules combiner can test millions of rules combinations and select the one with the best performance in minutes because it does this by combining

results instead of running each combination separately against the corpus. Figure 9.8 and Table 9.1 show the best combination produced by the ARDAKE Rules Combiner for 6 Population elementary rules. The combination has an F-Score of 35% while the highest F-Score for an elementary rule is 26%.

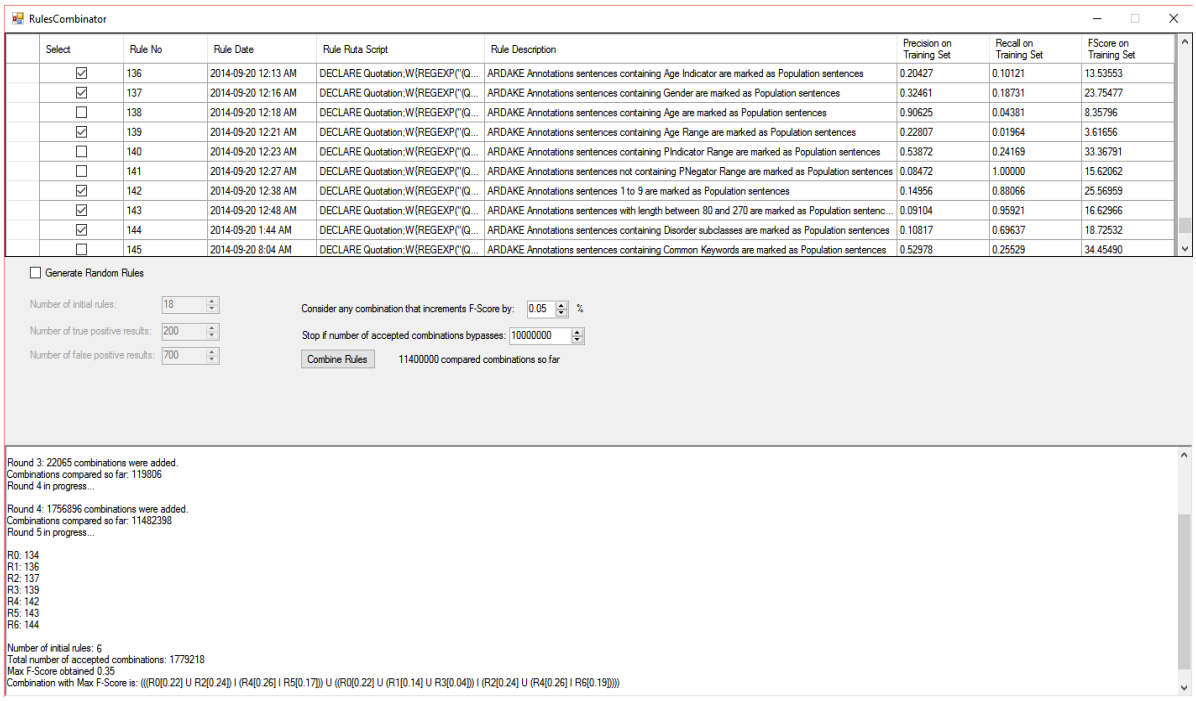


Figure 9.8: The ARDAKE Rules Combiner

Table 9.1: Output produced by the ARDAKE Rules Combiner

```
Round 1: 19 combinations were added.
Combinations compared so far: 84
Round 2 in progress...

Round 2: 231 combinations were added.
Combinations compared so far: 1072
Round 3 in progress...

Round 3: 22065 combinations were added.
Combinations compared so far: 119806
Round 4 in progress...

Round 4: 1756896 combinations were added.
Combinations compared so far: 11482398
Round 5 in progress...

R0: 134
R1: 136
R2: 137
```

```
R3: 139
R4: 142
R5: 143
R6: 144

Number of initial rules: 6
Total number of accepted combinations: 1779218
Max F-Score obtained 0.35
Combination with Max F-Score is: (((R0[0.22] U R2[0.24]) I (R4[0.26] I R5[0.17])) U ((R0[0.22] U (R1[0.14] U R3[0.04]))
I (R2[0.24] U (R4[0.26] I R6[0.19]))))
```

9.5 Benchmarking with Machine Learning Algorithms

It was quite simple, using our approach and tools, to bypass the annotation results obtained by most state-of-the-art annotation algorithms and tools used in the NICTA-PIBOSO contest. Note that there is still lots of room to refine our annotation rules and obtain even better annotation results.

Despite the fact that our rules did not take any advantage of the distinctive characteristics of structured versus unstructured abstracts, our F-Scores for structured and unstructured abstracts were higher than most state-of-the-art algorithms as show in Table 9.2.

Our Population F-Score for structured abstracts was 15% higher than the one obtained by [52] and 6% above the one obtained by [53] but 5% less than the best results obtained by [54]. Our F-Score for Population sentences in unstructured abstracts was 26% higher and more than doubled the one obtained by Verbeke et al. for the same classification. It was also 8% higher than the one obtained by Kim et al. but 11% lower than the Sarker et al. one.

Our Intervention results were even better as we had the highest F-Score for classifying Intervention sentences in structured abstracts. Our Intervention F-Score for the structured abstracts was 13% higher than the highest F-Score obtained by Sarker et al... This represents an increase 43% over the best F-Score obtained during the ALTA-NICTA PIBOSO contest. Our F-Score here is 17% higher than the Verbeke et al. and more than doubled the one obtained by Kim et al. For unstructured abstracts, our F-Score was almost the triple of the one obtained by Kim et al. and more than doubled the Verbeke et al. one and only 3% below the highest F-Score obtained by Sarker et al.

Table 9.2: Population and Intervention annotation results

PIBOSO Terms	Sarker et al.		Kim et al.		Verbeke et al.		Our approach	
	S	U	S	U	S	U	S	U
Population	0.45	**0.59**	**0.56**	0.40	0.36	0.22	0.51	0.48
Intervention	0.30	**0.39**	0.20	0.13	0.26	0.16	**0.43**	0.36

While we use the NICTA-PIBSO competition results as a benchmark, it is important to note that a study done by [55], after the competition, produced better results using a ML approach based on a discriminative set of features from the PIBOSO corpus. The latter solution produced results that are much lower than other state-of-the-art when tested on balanced PIBOSO corpora generated using various data balancing strategies [51].

9.6 Chapter Summary

We used the tools we developed and described in previous chapters to evaluate and optimize our elementary KE rules then to automatically generate the rules combination that produced better results than those obtained by most state-of-the-art KE algorithms and tools when run over the NICTA-PIBOSO corpus to identify population and intervention sentences. Our tools can be used, in the same manner, in other rule-based KE projects to optimize rules and generate the best rules combination.

CHAPTER 10

Conclusion

10.1 Fulfillment of Research Objectives

We set our research objectives in Table 3.1 based on KEFUD research challenges described in Table 2.4 and showed how they were all met throughout the chapters of this thesis. We summarize everything in Table 10.1 and provide references to the sections containing the proof and details about the fulfillment of each objective.

Table 10.1: Fulfillment of Research Objectives

Research Objective	**Targeted Challenge**	**Fulfillment**
Simplify the creation and maintenance of KEFUD rules.	Creating and maintaining KEFUD rules	A significant part of our research and work was to meet this objective. We designed and implemented ARDAKE (CHAPTER 5) that allows users to create and modify KEFUD rules in a simple, visual, and consistent way. We used ARDAKE in CHAPTER 8 to show how simple it is to create KEFUD rules that produce results that are comparable to, and in some cases better than, those obtained by state-of-the-art ML algorithms as shown in Table 9.2.
Make it easy for users, particularly domain experts, to rely on ontology concepts and relationships while	Creating semantic rules based on ontologies	In order to compensate for the lack of semantic rules in UIMA Ruta, most Ruta extensions we created in 5.6 were semantic-based. We showed in CHAPTER 8 that using ARDAKE to

creating their KEFUD rules.		create Semantic rules based on ontologies is similar to the creation of all other ARDAKE rule types. It is done using a simple, visual user interface allowing users to select desired ontologies and concepts with few mouse clicks. See Figure 8.5 for an example of a semantic rule in ARDAKE.
Explore the full potential of n-grams including their positive, negative, and collective correlation with the patterns of interest in order to get a better KEFUD performance.	Determining the right set of n-grams to use for KEFUD rules	We dedicated CHAPTER 4 to study the importance and the limitations of n-grams. We proposed an algorithm in 4.5 to generate n-gram decision trees based on positive and negative correlations between n-grams. We discussed in 4.6 how n-gram decision trees can be used to improve the quality of KEFUD rules. We also created the Corpus Analyser (5.3.4) to automatically generate KEFUD rules based on our n-gram decision trees. In Section 9.3.3 we showed that KEFUD rules generated based our n-gram decision trees produce high F-Scores when the training set properly represents the domain data. The same rules produce very low F-Scores when the training set does not properly represent the domain data.

Find a simple, accurate, and efficient way to identify the best combination of KEFUD rules.	Finding the best combination of KEFUD rules	In Section 5.3.8, we showed how the Rules Combiner compares millions of rules combinations and select the best one in seconds. We did may simulations and noticed that the rules combinations produced by our Rules Combiner have F-Scores that are significantly higher than those of the initial rules.
Make it trivial for rule designers to correct a failing rule in either matching true positive results or avoiding false positive results.	Evaluating and optimizing KEFUD rules	The Rules Results Analyser described in 5.3.7 shows the results of each rule using a pie chart with different colors to depict TP, FP, TN, and FN. It gives an immediate insight on the quality of the results and suggests whether the rule condition(s) should be tightened or relaxed in order to improve the rule performance.

10.2 Research Contributions

Our research was concentrated on the creation of simple, yet efficient, rules for knowledge extraction from unstructured data. Our goal was to make it possible for non-technical domain experts to create, maintain, and run knowledge extraction while reducing the need for business analysts, data engineers, and software engineers and minimizing the risk of lost or incorrect knowledge due to miscommunication amongst resources with different backgrounds and skillsets.

We designed an architecture for KEFUD and developed a prototype named Adaptive Rules-Driven Architecture for Knowledge Extraction (ARDAKE), providing an interface for domain experts to develop text mining syntactical and semantic rules. These are then executed, scored,

and combined into pipelines of analysis engines using an open source NLP backend, integrated with a graph database and rules language. Our results visualizer gives an immediate insight into rules performance and how to optimize individual rules based on their results.

We also defined and implemented an algorithm to automatically generate n-grams based annotation rules that ARDAKE users can add to their rules set. Our results-based rules combiner efficiently compares the performance of thousands of rules combinations without having to execute these combinations over the corpus. Once the optimal combination of rules is found and recommended by the system, it can then be applied to annotate text corpora.

We defined a number of semantic and non-semantic rule elements (Patterns, Conditions, and Actions) and implemented them as UIMA Ruta extensions. We then extended ARDAKE to make it possible for users to use our rule elements while creating or modifying theirs rules using the ARDAKE Visual Rules Editor.

We demonstrated the performance of our prototype, algorithm, rule extensions, and tools using a text corpus that had been extensively studied with machine learning. We imported the PIBOSO dataset and successfully improved sentence classification in a set of 1000 abstract from the medical literature focused on a single disease (i.e., spinal cord injury). Our results outperformed those obtained by most state-of-the-art ML algorithms used in the ALTA-NICTA PIBOSO contest confirming the simplicity and the effectiveness of our approach as well as the potential of integrating text mining and semantic annotation.

10.3 Significance

Our results offer the first published contribution toward extending the Ruta rules language to integrate semantic and NLP technologies. Most NLP and semantic platforms are separated, with the rare exception of the General Architecture for Text Engineering (GATE) platform that allows integrating with Graph DB and writing rules in Java. We showed that our environment can offer the same functionality with great simplicity and expandability. As well, the Service Component Architecture (SCA) environment provided by UIMA provides a much greater

flexibility and reliability in high-performance, distributed, and real-time enterprise environments.

We also made the first extensive use of n-grams to compose, test, and optimize rules to be integrated within a semantic engine. Our approach contrasts with traditional machine learning applications in text mining. Instead of looking at text with only Part-of-Speech tagging, n-grams are a convenient middle-ground between, on one hand, the indiscriminate bag-of-words and TD-IDF computation, and on the other hand, the computationally complex formalism of an ontology. Our algorithm takes the advantage of the quantitative as well as qualitative aspects of n-grams, which encompass in part the logic behind text associations, while keeping the relevance of frequency distributions of such combinations. The nested capability of our algorithm also goes well beyond similar efforts in hierarchical text mining.

The prototype we developed is also among the first attempts to making NLP and semantic rules integration most user-friendly. We ensured that our interfaces take in consideration the literacy level of end-users, and exploit the simplicity and expressivity of ontologies to document a knowledge base. We were also very faithful to the logic of the Ruta language, by representing its very elements and structure in our visual rules development process. We also provided great transparency by allowing end-users to check the "back-end" of our code implementation, and verify the code produced of these rules, as well as their validity and reliability.

We proposed a new way of visualizing the quality performance of rules, extending the already well-known classification performance indicators. While recognizing that traditional indicators are highly meaningful, we wanted to make them more accessible to non-technical end-users. As such, a visual and colorful representation, while ensuring a limited set of elements and relationships, became a valuable asset to communicate classification results to domain experts. We hope this particular contribution will find extensive use and relevance in a variety of fields where these traditional performance indicators are commonly used.

Finally, we contributed to the EBM PICO literature by improving the classification of 2 key classes for medical literature abstracts. This research area has mostly been studied using machine learning, as demonstrated in the exclusive use of such technologies in analyzing the PIBOSO corpus. In addition to attempting to innovate in service to the biomed informatics profession, we provided hope to push ever further the performance and quality of classification engines in the medical profession, something that may have been viewed until recently infeasible given the high complexity of these scientific disciplines. We are confident that our approach can help EBM PICO researchers reach much greater success, allowing them to tackle text corpora and knowledge bases of greater complexity than those studied so far in the literature.

10.4 Limitations

One of the limitations of ARDAKE is that it is a windows-based application which requires installing it on a windows operating system before it can be used. Our future web-based implementation of ARDAKE will eliminate the need to install it making it available to more users.

Like other rule-based KE solutions, ARDAKE suffers an inherit limitation for analysing poor quality data with erroneous and/or missing values. This can be solved using data cleansing or by creating extra rules to handle specific cases of erroneous or missing data.

Creating and managing a large number of rules is a challenge for any rule-based KE solution. This limitation can be reduced by grouping related rules into libraries and allowing users to search and import rules from those libraries.

Another limitation is the lack of integration with well known text annotation platforms such as GATE. ARDAKE supports the conversion of its rules into the UIMA Ruta rules language. The integration with GATE and it rules language JAPE can be accomplished using the UIMA-GATE interoperability layer provided by GATE.

We must also consider the risks of diversifying the use of our prototype to text corpora and application domains of greater complexity. We have not yet tested our tool with scenarios of vast and deep relationships in complex ontologies. We have also not yet studied the challenging interpretations when n-grams must be combined with numeric data and symbols.

Finally, our test has been highly focused on improving the classification performance of a specific corpus. While ARDAKE offers evident advantages relative to other platforms, we still need to measure the tangible impact on the task efficiency and efficacy of end-users in a variety of application domains. This would require a more behavioral analysis of a richer and more dynamic rules-driven decision-making environment.

10.5 Application Potential

We are undoubtedly entering the era of unstructured information, and text and rules will become the next frontier in Big Data, Analytics, and Intelligent Systems research. However, the integration of NLP and semantic technologies is still in its infancy. We therefore have very limited hints as to the potential of using our approach, other than those presently revealed by existing uses of text mining.

Yet we can envision a future in a variety of industries, organizations, professions, sectors, and even any human activity, where any end-user (no longer simply domain experts) will be able to seamlessly and instinctively “think in terms of adaptive rules” in decision-making.

Our vision for a future making text the core element of any decisions with humans in the loop may, possibly, revolutionize our conception of intelligent systems. As opposed to always being conceived “in-support-of” human activity, the greater facility to integrate text directly within intelligent information processing may create true cyborg intelligence, where humans and machines co-depend on one another for the endless virtuous cycle of knowledge extraction, knowledge combination, knowledge creation, knowledge codification, knowledge diffusion, and knowledge reuse.

10.6 Related Works

A recent study was done by [56] to compare the most popular text mining tools. Fifty-five text mining tools were identified out of which thirty-nine are proprietary and thirteen are open source and only three are web-based tools that offers very limited functionality. The study listed the techniques and features supported by each tool. While some tools include features that are not currently supported by ARDAKE such as data cleansing text summarization, and audio/video content analysis, the rule-based tools are almost entirely based on writing rules using DM or rule languages like R, NLP++, and GATE Jape. For instance, VisualText can auto-generate some rules if users provide sufficient text annotations of specific text portions such as phone numbers but users must write NLP++ code with complex syntax for more advanced rules. In addition to statistical and other machine learning techniques, OdinText has limited support for rule-based solutions and its rules are proprietary and not reusable by other tools or for other corpora [57]. ARDAKE rules can be exported into standard UIMA Ruta rules and can therefore be reused by any UIMA-based tool. Tovek Tools offer a query editor allowing users to create IE rules and combine them using Boolean expressions. Their query editor is intended for expert users who need to write their rules in full-text using a specific rule syntax.

10.7 Future Research

Inspired by the diversity of potential application areas, our technical research program will encompass both infrastructure and application development.

On the software-side, we will continue with the aim of improving our prototype and its customization to various domains. In addition to overcoming the known ARDAKE limitations, our future research will focus especially on the detection of temporal events and complex relationships extraction. We also plan to research more enterprise-grade environments, such as those depending on the creation of business rules engine that can be used to automate business processes and data driven decision making.

However, we envision most of our efforts, and especially research collaborations, to address the application-side of our field. As a starting point, organizational decision-making will

greatly benefit from knowledge extraction and modeling that integrates concepts and relationships annotations created by the execution of ARDAKE rules over domain corpora. As well, our research agenda will increasingly integrate multidisciplinary and multilingual text corpora, to accurately reflect the changing workplace and globalization. Great consideration for combining quantitative and qualitative data should also offer significant research challenges, many of which will also reflect the increasing complexity of diverse applications of data and text mining by workers of various professionals.

Finally, as we did in the field of EBM PICO, we hope to make further contributions to various domains of the arts, humanities, management, health and natural sciences where semantic text mining is becoming increasingly valuable. We will continue our efforts in developing the mission of the informatics profession in becoming partners and agents of change in every aspect of human knowledge, hence requiring ever greater care for end-user friendliness and empowerment in using intelligent systems for knowledge management.

ANNEX I

Ruta Script Generated by ARDAKE for Population Annotations

```
DECLARE Quotation;
W{REGEXP("((QUOTE))")->MARK(Quotation)};

DECLARE Sentence;
Quotation # Quotation{->MARK(Sentence, 2)};

Sentence {Length(0, 10) ->UNMARK(Sentence)};

DECLARE AgeIndicator;
WORDLIST AgeIndicator = 'AgeIndicator.txt';
Document{->MARKFAST(AgeIndicator, AgeIndicator)};

DECLARE AgeKeyword;
WORDLIST AgeKeyword = 'AgeKeyword.txt';
Document{->MARKFAST(AgeKeyword, AgeKeyword)};

DECLARE AgeUnit;
WORDLIST AgeUnit = 'AgeUnit.txt';
Document{->MARKFAST(AgeUnit, AgeUnit)};

DECLARE Gender;
WORDLIST GenderList = 'GenderList.txt';
Document{->MARKFAST(Gender, GenderList)};

DECLARE Race;
WORDLIST RaceList = 'RaceList.txt';
Document{->MARKFAST(Race, RaceList)};

DECLARE LetterNumber;
WORDLIST LetterNumberList = 'LetterNumberList.txt';
Document{->MARKFAST(LetterNumber, LetterNumberList)};

DECLARE Number;
(NUM | LetterNumber){->MARK(Number)};

DECLARE NumberRange;
Number ANY Number{->MARK(NumberRange, 1, 3)};

DECLARE Age;
((Number AgeUnit) | (Number AgeUnit AgeKeyword) |  (Number ANY? AgeUnit
ANY? AgeKeyword)){->MARK(Age, 1, 2)};
//Number AgeUnit AgeKeyword{->MARK(Age, 1, 3)};
//Number ANY? AgeUnit ANY? AgeKeyword{->MARK(Age, 1, 5)};

DECLARE AgeRange;
"between" Number "and" Number{->MARK(AgeRange, 1, 4)};

("under" | "over") Number{->MARK(AgeRange, 1, 2)};
```

```
Number "and" ("under" | "over" | "older" | "younger"){->MARK(AgeRange, 1,
3)};

DECLARE PopulationNgram;
WORDLIST P_Indicators = 'P_Indicators.txt';
Document{->MARKFAST(PopulationNgram, P_Indicators)};

DECLARE PopulationKeyword;
WORDLIST PopulationKeywor_List = 'PopulationKeyword.txt';
Document{->MARKFAST(PopulationKeyword, PopulationKeywor_List)};

DECLARE NonPopulationNgram;
WORDLIST P_Negators = 'P_Negators.txt';
Document{->MARKFAST(NonPopulationNgram, P_Negators)};

DECLARE PIBOSO_Disease;
(W | (W W) | (W W W) | (W W W W) | (W W W W W)){OR(
SubClassOf("<http://bioonto.de/mesh.owl#C10.228.854>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#C10.228.140.199>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#A05.360.444>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#F01.145.126>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#C20.111.258>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#C10.314>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#D003123>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#D010190>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#D002277>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#E01.370.378.325>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#C26>", "mesh", true),
SubClassOf("<http://bioonto.de/mesh.owl#A08.186>", "mesh", true)) -
>MARK(PIBOSO_Disease)};

WORDLIST MeshConcept_List = 'MeshConcepts.txt';
Document{->MARKFAST(PIBOSO_Disease, MeshConcept_List)};


DECLARE PopulationSentenceCandidate;
Sentence{CONTAINS(PopulationNgram)->MARK(PopulationSentenceCandidate)};

Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bretrospective\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bretrospectively\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bprospective\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bQOL\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bobstruction\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\baged\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\brecruited\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bconducted\\b).*$") -
>MARK(PopulationSentenceCandidate)};
```

```
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\breferred\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bexamined\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\btreated\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bassessed\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bevaluated\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bstudied\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bselected\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bidentified\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\btested\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\binvestigated\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bmeasured\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bconsecutive\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\benrolled\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bsuffered\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\brandomized\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bcriteria\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bdiagnosed\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\b\\(N =\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bphysical\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bcompleted\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bstudy\\b)(?=.*\\bincluded\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bstudy\\b)(?=.*\\bexamined\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatient\\b)(?=.*\\bexperiences\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatient\\b)(?=.*\\bmuscle\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bsuffered\\b)(?=.*\\bfrom\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\beffect\\b)(?=.*\\bexamined\\b).*$") -
>MARK(PopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bsubjects\\b)(?=.*\\bscheduled\\b).*$") -
>MARK(PopulationSentenceCandidate)};
```

```
Sentence{REGEXP("^(?=.*\\bsubjects\\b)(?=.*\\binterview\\b).*$") -
>MARK(PopulationSentenceCandidate)};

DECLARE NonPopulationSentenceCandidate;
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bgroup\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\binjury\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bcare\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\brisk\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\boutcome\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bsymptoms\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
//Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bassociated\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bsevere\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\blevels\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bmean\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bgroups\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bCHF\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bresults\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bprimary\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bsurgery\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};
Sentence{REGEXP("^(?=.*\\bpatients\\b)(?=.*\\bpatient\\b).*$") -
>MARK(NonPopulationSentenceCandidate)};

DECLARE PrePopulationSentence;
PopulationSentenceCandidate {->MARKSCORE(50, PrePopulationSentence)};

DECLARE PopulationPosition;
Sentence {OR(POSITION(Document, 1), POSITION(Document, 2),
POSITION(Document, 3), POSITION(Document, 4), POSITION(Document, 5),
POSITION(Document, 6), POSITION(Document, 7), POSITION(Document, 8),
POSITION(Document, 9), POSITION(Document, 10)) -
>MARK(PopulationPosition)};

PopulationPosition{->MARKSCORE(10, PrePopulationSentence)};

Sentence {OR(CONTAINS(AgeIndicator), CONTAINS(Age), CONTAINS(AgeRange)) -
>MARKSCORE(20, PrePopulationSentence)};
Sentence {CONTAINS(Gender) ->MARKSCORE(20, PrePopulationSentence)};
Sentence {CONTAINS(Race) ->MARKSCORE(20, PrePopulationSentence)};
Sentence {CONTAINS(PopulationKeyword) ->MARKSCORE(20,
PrePopulationSentence)};
```

```
Sentence {CONTAINS(PIBOSO_Disease) ->MARKSCORE(20,
PrePopulationSentence)};
Sentence {Length(85, 265) -> MARKSCORE(10, PrePopulationSentence)};

PrePopulationSentence {CONTAINS(NonPopulationNgram)->MARKSCORE(-30,
PrePopulationSentence)};
NonPopulationSentenceCandidate {->MARKSCORE(-50, PrePopulationSentence)};
PrePopulationSentence {OR(Length(0, 84) , Length(266, 100000)) ->
MARKSCORE(-50, PrePopulationSentence)};
Sentence {AND(-POSITION(Document, 1), -POSITION(Document, 2), -
POSITION(Document, 3), -POSITION(Document, 4), -POSITION(Document, 5), -
POSITION(Document, 6), -POSITION(Document, 7), -POSITION(Document, 8), -
POSITION(Document, 9), -POSITION(Document, 10)) ->MARKSCORE(-50,
PrePopulationSentence)};

DECLARE PSentence;
PrePopulationSentence {SCORE(41, 1000000)->MARK(PSentence)};
PrePopulationSentence {->UNMARK(PrePopulationSentence)};
PopulationPosition {->UNMARK(PopulationPosition)};
Sentence {->UNMARK(Sentence)};
Quotation {->UNMARK(Quotation)};
```

BIBLIOGRAPHY

[1] Simpson, M. S., & Demner-Fushman, D., (2012), "Biomedical text mining: A survey of recent progress", ***Mining text data***, Springer, 465-517.

[2] Chiticariu, L., Li, Y., & Reiss, F. R., (2013), "Rule-based information extraction is dead! long live rule-based information extraction systems!", Paper presented at the ***EMNLP***: 827-832.

[3] Khan, N., Yaqoob, I., Hashem, I. A. T., Inayat, Z., Mahmoud Ali, W. K., Alam, M., Shiraz, M., & Gani, A., (2014), "Big data: survey, technologies, opportunities, and challenges", ***The Scientific World Journal***, 2014.

[4] Sharda, R., Delen, D., & Turban, E., (2013), ***Business Intelligence: A Managerial Perspective on Analytics***, Prentice Hall Press.

[5] Oren, E., Möller, K., Scerri, S., Handschuh, S., & Sintek, M., (2006), "What are semantic annotations", ***Relatório técnico. DERI Galway***.

[6] Gruber, T., (1993), "What is an Ontology".

[7] Berners-Lee, T., Hendler, J., & Lassila, O., (2001), "The semantic web", ***Scientific american***, 284 (5): 28-37.

[8] McIlraith, S. A., Son, T. C., & Zeng, H., (2001), "Semantic web services", ***IEEE intelligent systems***, 16 (2): 46-53.

[9] Paolucci, M., Kawamura, T., Payne, T. R., & Sycara, K., (2002), "Semantic matching of web services capabilities", ***The Semantic Web—ISWC 2002***, Springer, 333-347.

[10] Fawcett, T., (2006), "An introduction to ROC analysis", ***Pattern recognition letters***, 27 (8): 861-874.

[11] Sokolova, M., Japkowicz, N., & Szpakowicz, S., (2006), "Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation", ***AI 2006: Advances in Artificial Intelligence***, Springer, 1015-1021.

[12] Hand, D. J., & Till, R. J., (2001), "A simple generalisation of the area under the ROC curve for multiple class classification problems", ***Machine learning***, 45 (2): 171-186.

[13] Ling, C. X., Huang, J., & Zhang, H., (2003), "AUC: a better measure than accuracy in comparing learning algorithms", ***Advances in Artificial Intelligence***, Springer, 329-341.

[14] Devedzic, V., (2004), "Education and the semantic web", ***International Journal of Artificial Intelligence in Education***, 14 (2): 165-191.

[15] Grau, B. C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., & Sattler, U., (2008), "OWL 2: The next step for OWL", ***Web Semantics: Science, Services and Agents on the World Wide Web***, 6 (4): 309-322.

[16] Cunningham, H., (2002), "GATE, a general architecture for text engineering", ***Computers and the Humanities***, 36 (2): 223-254.

[17] Ferrucci, D., & Lally, A., (2004), "UIMA: an architectural approach to unstructured information processing in the corporate research environment", ***Natural Language Engineering***, 10 (3-4): 327-348.

[18] El-Kass, W., Gagnon, S., & Iglewski, M., (2012), "Adaptive Rules-Driven Architecture for Knowledge Extraction (ARDAKE) with UIMA", Paper presented at the ***International Conference on Electrical and Computer Systems (ICECS'12), Ottawa***: 15-27.

[19] Kluegl, P., Atzmueller, M., & Puppe, F., (2009), "TextMarker: A tool for rule-based information extraction", ***Proceedings of the Biennial GSCL Conference 2009, 2nd UIMA@GSCL Workshop***: 233-240.

[20] Goble, C., & De Roure, D., (2009), "The impact of workflow tools on data-centric research", ***Data Intensive Computing: The Fourth Paradigm of Scientific Discovery***: 137-145.

[21] Kano, Y., Dobson, P., Nakanishi, M., Tsujii, J., & Ananiadou, S., (2010), "Text mining meets workflow: Linking U-compare with Taverna", ***Bioinformatics***, 26 (19): 2486-2487.

[22] IBM, (2011), "LanguageWare Resource Workbench 7.2 - Create Parsing Rules", ***IBM Corporation***.

[23] Bank, M., & Schierle, M., (2012), "A Survey of Text Mining Architectures and the UIMA Standard", Paper presented at the ***LREC***: 3479-3486.

[24] Kluegl, P., Toepfer, M., Beck, P.-D., Fette, G., & Puppe, F., (2016), "UIMA Ruta: Rapid development of rule-based information extraction applications", ***Natural Language Engineering***, 22 (1): 1-40.

[25] Kotsiantis, S. B., Zaharakis, I., & Pintelas, P., (2007), "Supervised machine learning: A review of classification techniques".

[26] Chu, X., Morcos, J., Ilyas, I. F., Ouzzani, M., Papotti, P., Tang, N., & Ye, Y., (2015), "Katara: A data cleaning system powered by knowledge bases and crowdsourcing", Paper presented at the ***Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data***, ACM: 1247-1261.

[27] Hernández, M. A., & Stolfo, S. J., (1998), "Real-world data is dirty: Data cleansing and the merge/purge problem", ***Data mining and knowledge discovery***, 2 (1): 9-37.

[28] Krishnan, S., Haas, D., Franklin, M. J., & Wu, E., (2016), "Towards reliable interactive data cleaning: a user survey and recommendations", Paper presented at the ***HILDA@ SIGMOD***: 9.

[29] Rahm, E., & Do, H. H., (2000), "Data cleaning: Problems and current approaches", ***IEEE Data Eng. Bull.***, 23 (4): 3-13.

[30] Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R., (2000), "CRISP-DM 1.0 Step-by-step data mining guide".

[31] Galar, D., Kans, M., & Schmidt, B., (2016), "Big Data in Asset Management: Knowledge Discovery in Asset Data by the Means of Data Mining", Paper presented at the ***Proceedings of the 10th World Congress on Engineering Asset Management (WCEAM 2015)***, Springer: 161-171.

[32] Heimerl, F., Koch, S., Bosch, H., & Ertl, T., (2012), "Visual classifier training for text document retrieval", ***Visualization and Computer Graphics, IEEE Transactions on***, 18 (12): 2839-2848.

[33] Sánchez, J. S., Barandela, R., Marqués, A. I., Alejo, R., & Badenas, J., (2003), "Analysis of new techniques to obtain quality training sets", ***Pattern Recognition Letters***, 24 (7): 1015-1022.

[34] Elragal, A., & Haddara, M., (2014), "Big data analytics: A text mining-based literature analysis", Paper presented at the ***Norsk konferanse for organisasjoners bruk av IT***.

[35] Gupta, V., & Lehal, G. S., (2009), "A survey of text mining techniques and applications", ***Journal of emerging technologies in web intelligence***, 1 (1): 60-76.

[36] Langkilde, I., & Knight, K., (1998), "The practical value of n-grams in generation", Paper presented at the ***Proceedings of the ninth international workshop on natural language generation***, Citeseer: 248-255.

[37] Manning, C. D., & Schütze, H., (1999), ***Foundations of statistical natural language processing***, MIT Press.

[38] Shaoul, C., Westbury, C. F., & Baayen, H. R., (2013), "The subjective frequency of word n-grams", ***Psihologija***, 46 (4): 497-537.

[39] Schmid, H., (2013), "Probabilistic part-ofispeech tagging using decision trees", Paper presented at the ***New methods in language processing***, Routledge: 154.

[40] Boswell, D., (2003), "CSE 254 (Spring 2003)“Growing N-gram Trees for Language Modeling”".

[41] Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., & Lai, J. C., (1992), "Class-based n-gram models of natural language", ***Computational linguistics***, 18 (4): 467-479.

[42] Cheng, W., Greaves, C., & Warren, M., (2006), "From n-gram to skipgram to concgram", ***International journal of corpus linguistics***, 11 (4): 411-433.

[43] Feinerer, I., (2014), "Introduction to the tm Package Text Mining in R".

[44] El-Kass, W., Gagnon, S., & Iglewski, M., (2015), "A Visual and Results-Driven Rules Composition Approach for Better Information Extraction", ***IFAC-PapersOnLine***, 48 (3): 112-117.

[45] Harispe, S., Ranwez, S., Janaqi, S., & Montmain, J., (2014), "The semantic measures library and toolkit: fast computation of semantic similarity and relatedness using biomedical ontologies", ***Bioinformatics***, 30 (5): 740-742.

[46] Harispe, S., Ranwez, S., Janaqi, S., & Montmain, J., (2015), "Semantic Similarity from Natural Language and Ontology Analysis", ***Synthesis Lectures on Human Language Technologies***, 8 (1): 1-254.

[47] KABIRI, A., & CHIADMI, D., (2013), "SURVEY ON ETL PROCESSES", ***Journal of Theoretical & Applied Information Technology***, 54 (2).

[48] Pwee, K., (2004), "What is this thing called EBM?", ***Singapore medical journal***, 45 (9): 413.

[49] Richardson, W. S., Wilson, M. C., Nishikawa, J., & Hayward, R. S., (1995), "The well-built clinical question: a key to evidence-based decisions", ***Acp j club***, 123 (3): A12-13.

[50] Huang, X., Lin, J., & Demner-Fushman, D., (2006), "Evaluation of PICO as a knowledge representation for clinical questions", Paper presented at the ***AMIA annual symposium proceedings***, American Medical Informatics Association: 359.

[51] Hassanzadeh, H., Groza, T., Nguyen, A., & Hunter, J., (2014a), "Load Balancing for Imbalanced Data Sets: Classifying Scientific Artefacts for Evidence Based Medicine", ***PRICAI 2014: Trends in Artificial Intelligence***, Springer, 972-984.

[52] Verbeke, M., Van Asch, V., Morante, R., Frasconi, P., Daelemans, W., & De Raedt, L., (2012), "A statistical relational learning approach to identifying evidence based medicine categories", Paper presented at the ***Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning***, Association for Computational Linguistics: 579-589.

[53] Sarker, A., Mollá-Aliod, D., & Paris, C., (2013), "An approach for automatic multi-label classification of medical sentences".

[54] Kim, S. N., Martinez, D., Cavedon, L., & Yencken, L., (2011), "Automatic classification of sentences to support evidence based medicine", ***BMC bioinformatics***, 12 (2): S5.

[55] Hassanzadeh, H., Groza, T., & Hunter, J., (2014b), "Identifying scientific artefacts in biomedical literature: The evidence based medicine use case", ***Journal of biomedical informatics***, 49: 159-170.

[56] Kaur, A., & Chopra, D., (2016), "Comparison of text mining tools", Paper presented at the ***Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO), 2016 5th International Conference on***, IEEE: 186-192.

[57] Burita, L., & Halouzka, K., (2017), "The Effective Working with Tovek Tools", Paper presented at the ***Management Challenges in a Network Economy: Proceedings of the MakeLearn and TIIM International Conference 2017***, ToKnowPress: 185-194.