NTNU
Innovation and Creativity

Ontology Learning - Suggesting Associations from Text

Gøran Sveia Kvarv

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Jon Atle Gulla, IDI
Co-supervisor: Terje Brasethvik, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

With the introduction of semantic technologies in industry and government agencies, large-scale ontologies need to be constructed and maintained. Typically this work is done partly manually, but in most cases it would be too expensive to depend on manual labor for this process. What is needed is a tool set that helps us build ontologies efficiently and update them as the domain changes. Currently, the information system group at IDI at NTNU is experimenting with different tools to extract keywords and phrases from text.

What is lacking in the current tool set is a way of discovering relations between words and phrases from text. The goal of this project is to implement and evaluate an algorithm for such an association analysis that can suggest possible ontological relations. The algorithm should be evaluated against an existing ontology.

The association algorithm should build upon, or make use of, several of the existing tools. At a later stage, the tools should be integrated within an overall framework, for this particular project, we will experiment with using GATE - General Architecture for Text Engineering.

Assignment given: 19. January 2007
Supervisor: Jon Atle Gulla, IDI

Abstract

In many applications, large-scale ontologies have to be constructed and maintained. A manual construction of an ontology is a time consuming and resource demanding process, often involving some domain experts. It would therefore be beneficial to support this process with tools that automates the construction of an ontology.

This master thesis has examined the use of association rules for suggesting associations between words in text. In ontology learning, concepts are often extracted from domain specific text. Applying the association rules algorithm on the same text, the associations found can be used to discover candidate relations between concepts in an ontology. This algorithm has been implemented and integrated in GATE, a framework for natural language processing. Alongside the association rules algorithm, several information extraction and natural language processing techniques have been implemented, in which this algorithm is built upon. This has resulted in a framework for ontology learning.

A qualitative evaluation of the associations found by the system has shown that the associations found by the association rules algorithm has promising results for detecting relations between concepts in an ontology. It has also been found that this algorithm is dependent on an accurate extraction of keywords. Further, a subjective evaluation of GATE has shown that it is suited as a framework for ontology learning.

Preface

This report documents the work done in my master thesis at the Information Systems Group of the Department of Computer and Information Science, Faculty of Information Technology, Mathematics and Electrical Engineering at the Norwegian University of Science and Technology.

I would especially like to thank my supervisor Terje Brasethvik for useful feedback and comments during my work. I would also like to thank his companion, Anders Kofod-Petersen for his passionate discussions and comments. Further, I would like to express my gratitude to Professor Jon Atle Gulla for his guidance. Finally, I would like to thank my fellow master degree candidates Knut Vidar Siem, Martin Riegel, Claes Lyth Walsø and Hans Magnus Wold for their participation in the evaluation, and Anders Ganes for being a great motivator.

Trondheim, June 2007

Gøran Sveia Kvarv

Contents

1 Introduction **1**

1.1 Problem . . . 1

1.2 Objectives . . . 2

1.3 Assumptions . . . 2

1.4 Approach . . . 3

1.5 Report Structure . . . 4

2 Theory and Background **5**

2.1 Ontologies . . . 5

2.1.1 Relation Mechanisms in Ontologies . . . 6

2.2 Ontology Learning . . . 6

2.3 Information Extraction and Text Mining . . . 7

2.3.1 Natural Language Processing . . . 8

2.3.2 Keyword Extraction . . . 9

2.3.3 Cosine Similarity . . . 9

2.4 Framework for Natural Language Processing . . . 10

2.4.1 GATE . . . 10

3 Mining Associations from Text **13**

3.1 Association Rules . . . 13

3.2 Applying the Association Rules Algorithm on Text . . . 18

3.2.1 Structuring text . . . 18

3.2.2 Term Extraction and Filtering . . . 19

3.2.3 Textual Entities as Transactions 20
3.3 Related examples 20

4 Implementation 21
4.1 Architecture 21
4.2 Natural Language Processing Module 22
4.2.1 Lemmatizer 23
4.2.2 NounPhrase 25
4.3 Index Module 27
4.4 Association Rules Module 28
4.4.1 Generator 29
4.4.2 Rule 31
4.5 Ontology Module 33
4.5.1 Concept 33

5 Evaluation Method 35
5.1 Concepts 36
5.1.1 Evaluation Strategy 36
5.2 Associations 37
5.2.1 Evaluation Strategy 37
5.3 GATE 39

6 Evaluation Results 41
6.1 Concepts Evaluation Results 41
6.2 Associations Evaluation Results 44
6.2.1 Overlap between cosine similarity 44
6.2.2 Results for each group 45
6.3 GATE 61
6.4 Evaluation Summary 62

7 Conclusion 65

8 Further Work **67**

8.1 Concept Extraction 67

8.2 Building Concept Hierarchies 68

A Acronyms and Abbreviations **69**

B Building Concept Hierarchies **71**

C Digital Appendix **73**

References **77**

List of Figures

3.1 The process of finding association rules between terms in a document collection. 19

4.1 Architecture 22

4.2 NLP system 23

4.3 Sequence diagram, lemmatizer module 24

4.4 Screenshots of the initialization parameters of the lemmatizer 25

4.5 NounPhrase pseudo code 25

4.6 Screenshots of the initialization parameters of the noun phrase extractor 26

4.7 Screenshot of the resulting annotations from running the noun phrase component in GATE 27

4.8 Sequence diagram, association rules module 28

4.9 FrequentItemSetGenerator 29

4.10 FrequentItemSetGenerator 30

4.11 Pseudo code for generating rules 31

4.12 Screenshot of the initialization parameters of the association rules component. 32

4.13 Screenshot of the resulting association rules from running the association rules component in GATE. 33

4.14 Pseudo code for finding environments 34

5.1 A venn diagram of the groups resulting from the overlap between the related concepts found by the system and the cosine similarity method. 39

6.1 Concept evaluation, manually constructed ontology 42

6.2 Concept evaluation, semi-automatically constructed ontology 43

6.3 Cosine similarity overlap 45

6.4 Chart of the evaluated related concepts found by the system for each of the chosen concepts. 50

6.5 Chart of the evaluated related concepts found by cosine similarity for each of the chosen concepts. 56

6.6 Chart of the evaluated related concepts found by both the system and cosine similarity for each of the chosen concepts. 59

B.1 Concept Hierarchy pseudo code 71

List of Tables

3.1 The transaction database 14

3.2 Candidate item set, C_1 14

3.3 Frequent item set, L_1 15

3.4 Candidate item set, C_2 15

3.5 Frequent item set, L_2 16

3.6 Candidate item set, C_3 16

3.7 Frequent item set, L_3 16

3.8 Candidate item set, C_4 17

3.9 Frequent item set, L_4 17

3.10 The association rules produced by the apriori algorithm 18

5.1 Overview of the evaluation 36

5.2 Possible evaluation scores of related concepts 39

6.1 Related concepts found only by the system for the concept ”cost”. . . 46

6.2 Related concepts found only by the system for the concept ”control”. 47

6.3 Related concepts found only by the system for the concept ”scope”. . 48

6.4 Related concepts found only by the system for the concept ”risk”. . . 48

6.5 Related concepts found only by the system for the concept ”project plan”. 49

6.6 Related concepts found only by cosine similarity for the concept ”cost”. 51

6.7 Related concepts found only by cosine similarity for the concept ”control”. 52

6.8 Related concepts found only by cosine similarity for the concept ”scope”. 53

6.9 Related concepts found only by cosine similarity for the concept ”risk”. 54

6.10 Related concepts found only by cosine similarity for the concept ”project plan”. 55

6.11 Related concepts found by both the system and cosine similarity for the concept cost. 57

6.12 Related concepts found by both the system and cosine similarity for the concept control. 57

6.13 Related concepts found by both the system and cosine similarity for the concept scope. 58

6.14 Related concepts found by both the system and cosine similarity for the concept risk. 58

6.15 Related concepts found by both the system and cosine similarity for the concept project plan. 59

6.16 Group summary 60

Chapter 1

Introduction

At the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU) there has been substantial work in the field of semantic technologies. One of the researched areas is ontology learning. The goal of ontology learning is to automatically extract relevant concepts and relations from a document collection or other kinds of data sets. This master thesis is a part of the ongoing research in the field of ontology learning.

1.1 Problem

The following is a quote of the problem description:

> *"With the introduction of semantic technologies in industry and government agencies, large-scale ontologies need to be constructed and maintained. Typically this work is done partly manually, but in most cases it would be too expensive to depend on manual labor for this process. What is needed is a tool set that helps us build ontologies efficiently and update them as the domain changes. Currently, the information system group at IDI at NTNU is experimenting with different tools to extract keywords and phrases from text.*
>
> *What is lacking in the current tool set is a way of discovering relations between words and phrases from text. The goal of this project is to implement and evaluate an algorithm for such an association analysis that can suggest possible ontological relations. The algorithm should be evaluated against an existing ontology.*
>
> *The association algorithm should build upon, or make use of, several of the existing tools. At a later stage, the tools should be integrated within*

an overall framework, for this particular project, we will experiment with using GATE - General Architecture for Text Engineering."

The traditional ontology engineering approach is a time consuming process. It would therefore be beneficial to be able to automatically extract concepts and relations. At IDI, work has been made for automatic extraction of concepts.

Ontologies are explicit conceptualizations of a domain. The importance of ontologies has been recognized in fields and industries as diverse as semi-conductor manufacturing, aircraft design, enterprise process management, heterogeneous database integration, knowledge engineering and planning, to name a few. The role of ontologies is to capture domain knowledge and provide a commonly agreed upon understanding of a domain. The common vocabulary of an ontology, defining the meaning of terms and their relations, is usually organized in a taxonomy and contains modeling primitives such as concepts, relations, and axioms [Staab and Maedche, 2000].

This master thesis will take the work done a step further, and examine the discovery of relations between concepts extracted from text.

1.2 Objectives

With the problem description as a basis, three main objectives have been extracted:

1. Implement an association rules algorithm for suggesting associations from text.
2. Evaluate the results against existing ontology.

   As the associations found by the association rules algorithm can be used to discover relations between concepts in an ontology, it would be valuable to evaluate these associations against relations found in an ontology.
3. Evaluate the usability of GATE as an overall framework for ontology learning.

1.3 Assumptions

Based on the problem description and the objectives, the following assumptions are taken:

- The association rules algorithm should be integrated in GATE. As a consequence the association rules algorithm should be written in Java.

- Necessary tools that the association rules algorithm build upon, such as keyword extraction, will be provided.
- Ontologies based on the book "A Guide To The Project Management Body Of Knowledge" [PMI, 2004] and the first six chapters of "Organizational Project Management Maturity Model" [PMI, 2003], developed by Grimnes [2006] will be used for evaluation of the associations found.

1.4 Approach

A manual construction of an ontology is a time consuming and resource demanding process, often involving some domain experts. It would therefore be beneficial to support this process with tools that automates the construction of an ontology. In ontology learning, concepts are automatically extracted from domain specific text. Applying the association rules algorithm on the same text, the associations found can be used to discover candidate relations between concepts in an ontology. Previous research within the fields of information extraction and natural language processing constitutes a starting point. The association rules algorithm will build upon methods from these fields. The following approach will be carried out:

- Extract candidate keyphrases from the text

  The text will be processed with natural language processing components to find noun phrases.
- Weight and filter candidate keyphrases.

  The candidate keyphrases will be weighted and filtered according to some weighting scheme.
- Index the keyphrases

  The keyphrases will be indexed. That is, each keyphrases will have a mapping to each of the documents where the keyphrase occur. This index will be used by the association rules algorithm.
- Extract associations between the resulting keyphrases.

  The association rules algorithm will be used to discover associations between the keyphrases found in the text. By using domain specific text, these associations could be used to discover relations between concepts in an ontology within the same domain.

As the associations found by the association rules algorithm can be used to discover relations between concepts in an ontology, a qualitative evaluation of the associations found will be carried out.

1.5 Report Structure

The structure of this report is as follows:

- Chapter 2: Theory and Background

  This chapter introduces important theory and background for ontology learning. Relevant techniques of information extraction and natural language processing are presented.

- Chapter 3: Mining Associations from Text

  This chapter will further elaborate on the approach for finding associations from text.

- Chapter 4: Implementation

  This chapter presents the implementation of associations rules algorithm and other necessary components that the association rules algorithm is built upon.

- Chapter 5: Evaluation Method

  This chapter describes the chosen evaluation method for the evaluation of the associations found. It will in detail describe how the evaluation will be carried out.

- Chapter 6: Evaluation Results

  This chapter presents and discusses the evaluation results.

- Chapter 7: Conclusion

  This chapter presents concluding remarks for the work done.

- Chapter 8: Further Work

  This chapter discusses possible directions for further work. A technique for improving concept extraction is presented, alongside a method for building concept hierarchies.

Appendices:

- Appendix A: Acronyms and Abbreviations

- Appendix B: Building Concept Hierarchies

  This appendix contains pseudo code of how to use association rules to build concept hierarchies.

- Appendix C: Digital Appendix

Chapter 2

Theory and Background

At IDI at NTNU, substantial work has been made in the field of semantic technologies. One of the researched areas are ontology engineering, as described in Gulla [2006]. A manual construction of an ontology is a time consuming and resource demanding process, often involving some domain experts. It would therefore be beneficial to support this process with tools that as far as possible automates the construction of an ontology. This chapter introduces background and theory that is important in the field of ontology learning.

2.1 Ontologies

An ontology can be seen as a body of formally represented knowledge, a specification of a conceptualization. This means the objects, concepts and other entities that exist in an area of interest and the relationship among them [Gruber, 1995]. In a philosophical view this means the representation of what exist in the world. Brasethvik [2004] outlines two common causes for developing an ontology in information systems:

- To establish a shared understanding of a domain and thereby facilitate sharing of information therein.
- To enable the construction of intelligent or "semantic" applications.

The last item can be seen in context of Semantic Web, where there is a need of a representation of domain knowledge that allows machines to perform useful reasoning tasks on documents.

2.1.1 Relation Mechanisms in Ontologies

An ontology is often considered as an object model represented by a set of concepts that are taxonomically related by the transitive *ISA* relation and non-taxonomically related by a user named relation, for example *hasPart* [Maedche and Staab, 2000].

Web Ontology Language (OWL) is a language for creating ontologies and is one of the many recommendations provided by World Wide Web Consortium (W3C) related to Semantic Web. OWL is built to facilitate Extensible Markup Language (XML)'s ability to define customized tags and Resource Description Framework (RDF)'s flexible ability to represent data as objects and relationship between them. OWL has four important features that can be used to define relations [W3C, 2000]:

Class

A class defines a group of objects or concepts that belong together. Classes can be specialized by using subClassOf, as described below.

subClassOf

Stating that a Class is a subclass of another gives the ability to create hierarchies of classes that are either specializations or generalizations.

Property

Properties can be used to define relationships between objects or concepts. For a class Person, an example of a property is *hasChild*.

subPropertyOf

Hierarchies of properties can be made by stating that a property is a subproperty of one or more other properties. For a class Person that has a property *hasRelative*, an example of a subproperty can be *hasSibling*.

In addition properties can have stated characteristics and restrictions. Classes represent concepts that are taxonomically related while properties define non-taxonomically relationships between concepts.

2.2 Ontology Learning

A manual construction of an ontology is a time consuming and resource demanding process, often involving some domain experts. The reason for this is as Gulla [2006] points out, that the traditional ontology engineering approach involves:

- Form a team of ontology and domain experts.

- The ontology and domain experts model the domain with concepts and relationships.
- The ontology is tested against domain knowledge by the domain experts.
- The ontology experts verify the internal quality and application quality.

At IDI, there has been research in the field of ontology learning, supporting the traditional ontology engineering approach with tools that automatically extract candidate concepts from domain text for automatic ontology construction. Automatic keyphrase extraction was done by Borch [2005], where candidate keyphrase were extracted with a linguistic filter. The candidate keyphrases were then weighted and ranked, which yielded the final keyphrases. Grimnes [2006] constructed an ontology semi-automatically for Statoil, in the domain of project management. This was based on the work done by Borch [2005]. The extracted keyphrases were given to a domain expert who marked the keyphrases as irrelevant, ok or good. The keyphrases marked as irrelevant were removed and the resulting keyphrases were used to build the ontology. A manually constructed ontology was also created as a “gold standard” reference ontology. This ontology was built based on the traditional ontology engineering approach.

Finding non-taxonomic relations between concepts in an ontology is not a well-researched area [Maedche and Staab, 2000]. Association rules is a data mining technique that can be used to detect entities that co-occur. This has mostly been applied to databases. In this work, it will be tried to use association rules to look for words that co-occur in a document in a document collection. This could indicate a relation between those co-occurring words. In an ontology learning context, the association rules could provide suggestions of relations between concepts. What kind of relation this will be, will have to be provided manually in a later stage of the process. Association rules can therefore help the tradiontional ontology learning approach. Association rules will be described in detail in Chapter 3.

Tools for ontology learning relies heavily on the extraction of domain concepts (terminology) and categorization of these concept. Therefore both information extraction and text mining are important for an ontology learning system. The next section will describe some important information extraction and text mining techniques.

2.3 Information Extraction and Text Mining

The goal of information extraction is to extract structured information from unstructured text. Some subtasks of information extraction are among others, named entity recognition, co-reference and terminology extraction. Text mining is closely related to information extraction as it tries to find relevant patterns and trends.

However, text mining often involves the process of deriving linguistic features from the text and removal of words that are considered unimportant for mining. A typical task of text mining is categorization of the text.

2.3.1 Natural Language Processing

Manning and Shutze [2002] states that "The ultimate goal of research on Natural Language Processing is to parse and understand language." Natural Language Processing (NLP) is a subfield of linguistics that consists of automatic generation and understanding of natural human language. NLP systems analyze the text to find prahse structures and group words according to their syntactic and semantic type. Information extraction and text mining are often dependent of NLP. For example, text mining could be used to find relations between nouns in a text. This implies that the text is preprocessed by a Part-of-Speech tagger (see below) which groups the words in grammatical categories. The most important NLP components that will be used in this project are:

Tokenizer

Tokenization is usually one of the first steps of text processing. It divides the input text into units, called *tokens*. These tokens are either a word or something else like a number or a punctuation mark [Manning and Shutze, 2002].

Lemmatizer

A lemmatizer tries to find a word's *lemma*, its base form. For example the lemma of the word "processes" is "process". This is important in information extraction and text mining because "process" and "processes" would be considered two different words without lemmatization. Lemmatization is often achieved by looking up a given word in a dictionary or a lemma list. This is in contrast with the related process of stemming, where a word's stem is found by stripping off affixes, usually based on rules. Lemmatization is usually preferred in favor of stemming, because it is often difficult to derive the original word from a stem. An example of this is "manager" and "management", which both are reduced to the stem "manag".

Part-of-Speech Tagger

Tagging is a process that consists of labeling each word in a sentence with its appropriate *part of speech*, for example nouns, verbs, adjectives and prepositions. This is based on a predefined tagging scheme. One of the most common schemes for tagging is to train the tagger using an already tagged document collection, or *corpus*. This resolves word ambiguity that may appear using a simple approach like looking up a word in a dictionary to find its part-of-speech, since a word can belong to several morphological classes.

The most common approaches for training a tagger is rule-based and statistical models.

Phrase detection

Because languages have constraints on word order, grouping of words are organized in units called *phrases*. A noun phrase is a syntactic unit of a sentence where information about the noun, the *head*, is gathered. Noun phrases normally consist of zero or more adjectives, the noun and perhaps some prepositions. In a verb phrase on the other hand the verb is the head. The elements in a verb phrase are words that depend syntactically on the verb, except the noun [Manning and Shutze, 2002]. Phrase detection is often achieved by rules that define patterns of part-of-speech tags. Another approach is to use statistical measures where a phrase is defined as a sequence of words that co-occur frequently.

2.3.2 Keyword Extraction

Keywords in a document are words that describe the content of a document in a good way. Thereby, keywords can make important suggestions to concepts in an ontology when looking at a domain specific document collection. Keywords are often extracted by using a weighting scheme. One basic approach for term weighting is to use the term frequency (tf) in a document. This method just counts the number of times the term appears in a document. Usually the term frequency is normalized to prevent a higher term frequency in longer documents. The higher the term frequency, the more likely is it that the word captures important aspect of the content of a document. A more sophisticated scheme is to use the term frequency-inverse document frequency (tf-idf). If a term occurs frequently in a document, but infrequently in the document collection (corpus), a high weight will be assigned to that term in the document. The weight is therefore a statistical measure used to evaluate the importance of a term in a document in a corpus. This method consequently filters out terms that are common for the document collection [Manning and Shutze, 2002].

2.3.3 Cosine Similarity

In the same way as association rules, cosine similarity is a method that can be used to look for relations between entities. Cosine similarity is a measurement of similarity between any pair of words, that can be used for contextual categorization of words. From an ontology point of view this measure can provide relations between concepts, that means two concepts that are strongly connected. The words are represented as vectors in a multi-dimensional space. This is normally represented by a word-by-word matrix, where each entry in the matrix contains the number of times word i

co-occur with word *j*. Similarity between two vectors are found by evaluating the angle between the two vectors. The less the angle, the higher the similarity. The cosine similarity measure can be calculated as follows [Manning and Shutze, 2002]:

$$cos(\vec{x}, \vec{y}) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

2.4 Framework for Natural Language Processing

For different document collections, different situations and different purposes, different analyses will be relevant. It is thus desirable to place the association rules analysis in an environment or a framework that can nest all the required processing components (analyses) in various sequences, where each component uses the result from previous executed components.

Earlier work at IDI at NTNU has provided a set of components, often implemented in independent and stand-alone applications, written in several different programming languages. There has also been an attempt to create a workbench for such analyses with a protocol for communication between components and a shared file format to gather results [Brasethvik, 2004].

2.4.1 GATE

Genreal Architecture for Text Engineering (GATE) is a framework for text analysis developed in Java, available as open-source software [Gaizauskas et al., 1996]. The project description states that the association rules analysis should be developed as a component (plug-in) in GATE.

GATE is not only a framework for text engineering, it is also an architecture and a development environment. As an architecture, it defines the organisation of a text engineering system and assignment of responsibilities to different components. As a framework, it provides a set of built-in processing components that can be used, extended and customized according to specific needs. As a development environment it provides simple building blocks that developers can use to build new modules (plug-ins). This is facilitated through GATE's component based model and the GATE Application Programming Interface (API) [Cunningham et al., 2002].

The provided NLP resources in GATE is tied together in a package called ANNIE, Nearly-New Information Extraction System. These resources can be used as one unit or used as individual components along with others. Through GATE's graphical user interface the user can choose which components to execute and in what order they will be executed. ANNIE consists of the following processing components for english text:

- Tokenizer
- Sentence splitter
- Part-of-Speech tagger
- Gazetteer
- Semantic tagger
- Orthomatcher
- Coreferencer

By running these components, a document or a corpus can be annotated and stored. By loading an already annotated corpus the user can use the annotations for further processing whenever he or she wants. In addition GATE comes with a large set of plug-ins that can be loaded at any time. These include among others:

- Ontology Editor
- Machine Learning component
- WordNet component
- Information Retrieval component
- Stemmer with support for several languages
- Noun Phrase Chunker
- TreeTagger, another Part-of-Speech tagger with support for several languages

Chapter 3

Mining Associations from Text

This chapter describes a method for discovering associations between terms in a document collection, by means of *association rules*. First the association rules algorithm will be introduced and then it will be described how to apply this algorithm on text in a document collection.

3.1 Association Rules

Association rules is a technique in data mining that are used to discover elements that co-occur frequently within a dataset. Association rules were first introduced in Agrawal et al. [1993], as a technique for market basket analysis, that is predict the purchase behavior of customers. This was primarily done for a large database of items purchased on per-transaction basis. An example of such an association rule is the statement that 90% of the transactions that purchased bread and butter also purchased milk.

A formal statement of the problem of mining association rules is stated in Agrawal and Srikant [1994]:

Let I be a set of literals, called items.

Let D be a set of transactions where each transaction T is a set of items such that $T \subseteq I$.

A transaction T contains X, a set of some items in I, if $X \subseteq T$.

An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$ and $X \cap Y = \emptyset$.

A rule $X \Rightarrow Y$ holds in the transaction set D with *confidence* c if $c\%$ of the transactions in D that contain X also contain Y.

The rule$X \Rightarrow Y$ has *support* s in the transaction set D if $s\%$ of the transactions in D contain $X \cup Y$.

The idea is to generate all association rules that have support and confidence greater than a user specified minimum support and minimum confidence. The most important algorithm for the generation of association rules is the Apriori algorithm, introduced in Agrawal et al. [1993]. The algorithm finds all sets of items that have support greater than the minimum support. These sets are called *frequent item sets*. The association rules are then generated from these sets. The algorithm is illustrated in the following section.

Example

Table 3.1 shows a transaction database with the items A, B, C, D and E. A 1 indicates that the transaction with the given transaction id (TID) contains the item. A 0 indicates that item is not contained in the transaction.

TID	A	B	C	D	E
T1	1	1	1	0	0
T2	1	1	1	1	1
T3	1	0	1	1	0
T4	1	0	1	1	1
T4	1	1	1	1	0

Table 3.1: The transaction database

Given that the minimum support is 40%. In the first pass of the algortihm, it generates a candidate item set where the support for each set is found by counting the number of transactions that contains the given item set. The candidate item set, C_1 is shown in Table 3.2.

Itemset X	support(X)
A	100%
B	60%
C	100%
D	80%
E	40%

Table 3.2: Candidate item set, C_1

Since the minimum support is 40%, none of the item sets in C_1 are pruned. So the frequent item set, L_1 shown in Table 3.3, is the same as C_1.

Itemset X	support(X)
A	100%
B	60%
C	100%
D	80%
E	40%

Table 3.3: Frequent item set, L_1

In pass k the frequent item set, L_{k-1}, is joined by itself to produce the candidate item set C_k, where the first *k-2* in L_{k-1} must match. In pass 2, k - 2 = 0, so every item set in L_1 are joined with the others. This produces C_2 as shown in Table 3.4.

Itemset X	support(X)
A,B	60%
A,C	100%
A,D	80%
A,E	40%
B,C	60%
B,D	40%
B,E	20%
C,D	80%
C,E	40%
D,E	40%

Table 3.4: Candidate item set, C_2

Since the item set {B,E} has a support of 20% it is pruned, which results in the frequent item set, L_2 shown in Table 3.5.

Itemset X	support(X)
A,B	60%
A,C	100%
A,D	80%
A,E	40%
B,C	60%
B,D	40%
C,D	80%
C,E	40%
D,E	40%

Table 3.5: Frequent item set, L_2

In pass 3, the first item in the item sets must match, and these are joined. C_3 is shown in Table 3.6.

Itemset X	support(X)
A,B,C	60%
A,B,D	40%
A,B,E	20%
A,C,D	80%
A,C,E	40%
A,D,E	40%
B,C,D	40%
C,D,E	40%

Table 3.6: Candidate item set, C_3

The item set {A,B,E} is pruned. Table 3.7 shows the resulting L_3.

Itemset X	support(X)
A,B,C	60%
A,B,D	40%
A,C,D	80%
A,C,E	40%
A,D,E	40%
B,C,D	40%
C,D,E	40%

Table 3.7: Frequent item set, L_3

In pass 4 the two first item in the item sets must match and these are joined. C_4 is shown in Table 3.8.

Itemset X	support(X)
A,B,C,D	40%
A,C,D,E	40%

Table 3.8: Candidate item set, C_4

Both item sets satisfies the minimum support of 40%. This results in L_4, shown in Table 3.9.

Itemset X	support(X)
A,B,C,D	40%
A,C,D,E	40%

Table 3.9: Frequent item set, L_4

The algorithm terminates in pass 5 because the first three items in the two item sets doesn't match.

The algorithm now generates all rules. For every itemset l in frequent itemset, L_k, it finds subsets of size *k-1*. For every subset X, it produces a rule $X \Rightarrow Y$, where $Y = l$ - X. The rule is kept if the confidence,

$$\frac{support(X \cup Y)}{support(X)} \tag{3.1}$$

is greater than or equal to the minimum confidence. For a minimum confidence of 80%, the rules in Table 3.10 are generated:

A,B,D -> C	D,E -> A
B,C,D -> A	B,D -> C
A,C,E -> D	C,E -> D
A,D,E -> C	D,E -> C
C,D,E -> A	B -> A
A,B -> C	A -> C
B,C -> A	C -> A
B,D -> A	D -> A
A,D -> C	E -> A
C,D -> A	B -> C
A,E -> C	D -> C)
C,E -> A	E -> C
A,E -> D	E -> D

Table 3.10: The association rules produced by the apriori algorithm

3.2 Applying the Association Rules Algorithm on Text

To apply the association rules algorithm on text, the text has to be adapted and structured in a way that resembles a transaction with items in a database. Data mining in a database is totally different from text mining because a database deals with structured data, whereas text deals with unstructured data with special characteristics. Another difference is the large amount of items (terms) that is found in text in contrast to a retail database. The process of finding association rules from text is illustrated in Figure 3.1.

3.2.1 Structuring text

Delgado et al. [2002b] states that

> "The structure should reflect the way in which the user conceptualize the domain that is described by the data."

To find relations between terms in a document collection, the text needs to be structured in some way. This is normally considered the first step in text mining [Delgado et al., 2002b]. Structuring text benefits from the existing techniques of NLP, as described in Section 2.3.1. An important aspect of text structuring is to process the text with a Part-of-Speech tagger. This groups the words in syntactic

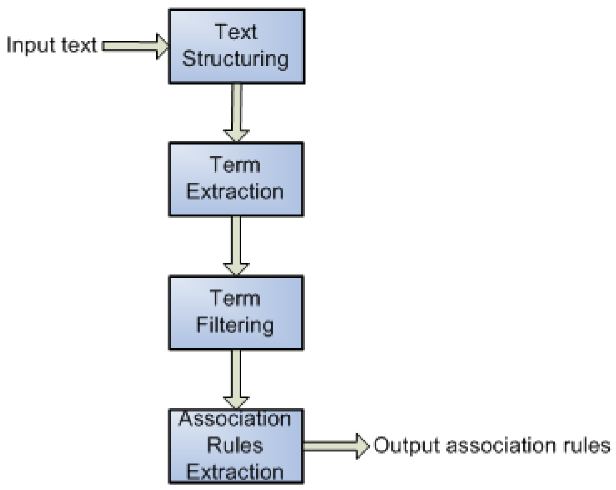


Figure 3.1: The process of finding association rules between terms in a document collection.

categories which in turn achieves structure. The Part-of-Speech tagger has another important meaning in text mining, namely the extraction of terms to mine. In most cases it is not desirable to extract relations using all the words in the text. Rather, it is desirable to mine for example nouns, noun phrases or keywords. In Haddad et al. [2000] it was experimented with using all the terms in the document collection. The relations discovered consisted of much noise and were not of much interest. Instead the choice fell on nouns because nouns best capture the semantics of a document.

3.2.2 Term Extraction and Filtering

To extract relations between all words in a document collection would be too time consuming and resource demanding, and results in too much noise as described in Section 3.2.1. Another approach is to use keywords. Because nouns capture the semantics of a document in a great way, these are generally used as candidate keywords. However, these terms are then weighted and filtered using some weighting scheme, as described in Section 2.3.2.

Term filtering is often used in conjunction with lemmatization as described in Section 2.3.1. Lemmatization prevents that words like “process” and “processes” are counted as two unique terms, by using their lemma instead. This is of most importance when applying weighting schemes like tf and tf-idf to extract keywords.

3.2.3 Textual Entities as Transactions

The most common area of application for association rules is to mine a large database with items contained in transactions. However textual documents do not have any transactions. The question is then how to apply textual entities as transactions. A number of possibilities exist, for example:

- Sentences
- Paragraphs
- Documents

For a large corpus, paragraphs and especially sentences will cause a large amount of transactions. Both Haddad et al. [2000] and Delgado et al. [2002a] argue that the use of documents as transactions give the best results.

3.3 Related examples

Delgado et al. [2002a] presented a system for query refinement that incorporated text mining. When a user tries to express a query, the terms provided are often not very specific because of the lack of domain background knowledge or the fact that other relevant terms for the query have not yet come to the users' mind. They used text mining, by the use of association rules to add relevant terms to the query, by analyzing documents retrieved by the initial query.

Words in the retrieved documents were weighted by the tf-idf and an initial set of terms were extracted. Stop words, that is words that doesn't have any useful meaning, were removed and the remaining terms were stemmed. By representing a transaction as the keywords in each document, they then executed the association rules algorithm and added relevant words to the query.

Nørvåg et al. [2006] presented a system for mining association rules from text where terms were extracted by including only nouns. Stop words were then removed and the resulting terms were stemmed. The terms were weighted with tf-idf and the k top-ranked terms were kept. Association rules were then mined on the basis of the resulting terms.

Chapter 4

Implementation

This chapter describes the implementation of the framework for ontology learning. As stated in the problem description, the information system group at IDI is experimenting with different tools to extract keywords and phrases. The association algorithm should build upon, or make use of several of the existing tools. However, as it turned out, the provided tools were implemented independently as stand-alone applications written in several different languages, without facilitation of reuse, some of them were not even available. As the association algorithm was to be integrated in GATE, written in Java, the tools provided were of no use. It was therefore necessary to reimplement these tools in a way that made them applicable for integration in a framework such as GATE. The source code for the implementation can be found in Appendix C.

4.1 Architecture

Figure 4.1 shows the architecture of the implemented framework. Each module is described in the next sections.

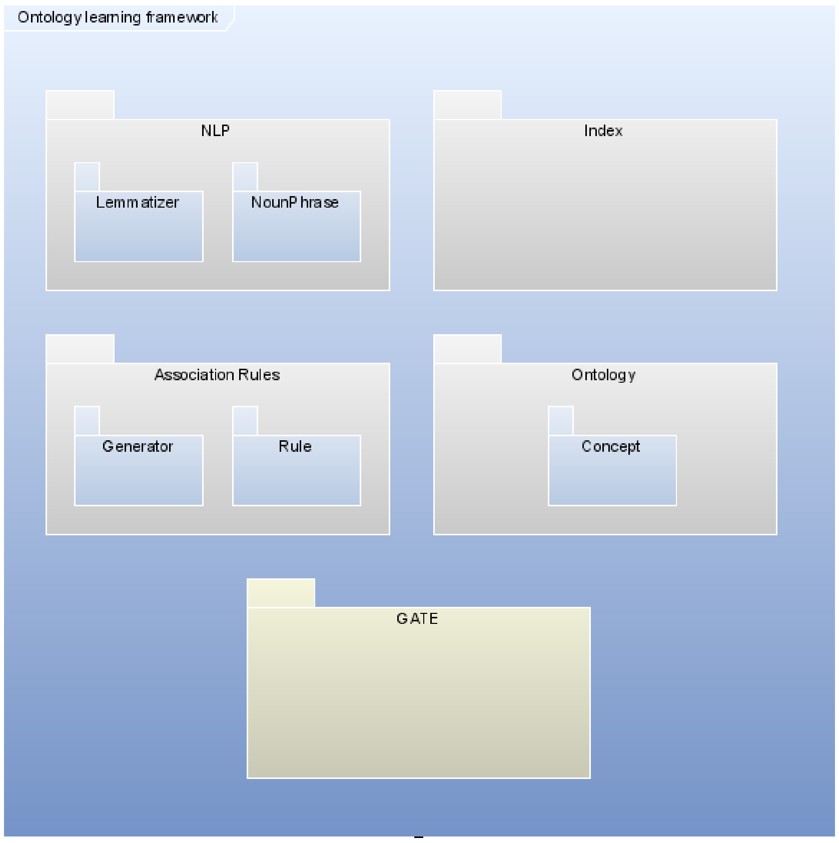


Figure 4.1: An overview of the implemented framework

4.2 Natural Language Processing Module

The natural language processing module is used in conjunction with GATE's built-in NLP components. It consists of a noun phrase extractor and a lemmatizer. Together with the GATE NLP components, they form a NLP system as visualized in Figure 4.2.

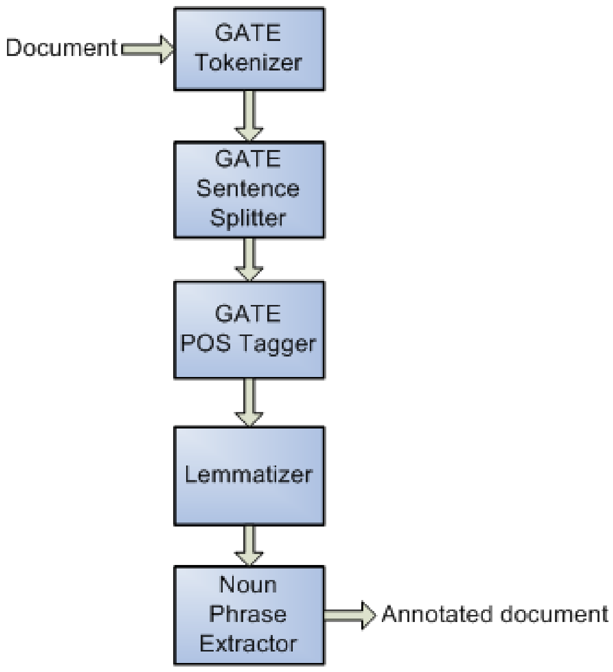


Figure 4.2: The NLP system comprising of GATE's NLP components and the implemented NLP module

Each document in a corpus is analyzed and the components of the NLP system adds its output as annotations to the document.

4.2.1 Lemmatizer

The lemmatizer module is responsible for finding the lemma of each word in a document. This is done by looking up the word in a lemma list, which contains a list of words and their associated lemma, made by Someya [1998]. The lemma list is found in Appendix C. If the current word looked up is not found in the lemma list, the lemma is set to the word as is.

The following java classes are contained in the lemmatizer module:

- EnglishDictionary
- EnglishLemmatizer
- Word
- WordNotFoundException

Figure 4.3 shows a sequence diagram of how the EnglishLemmatizer finds a word's lemma.

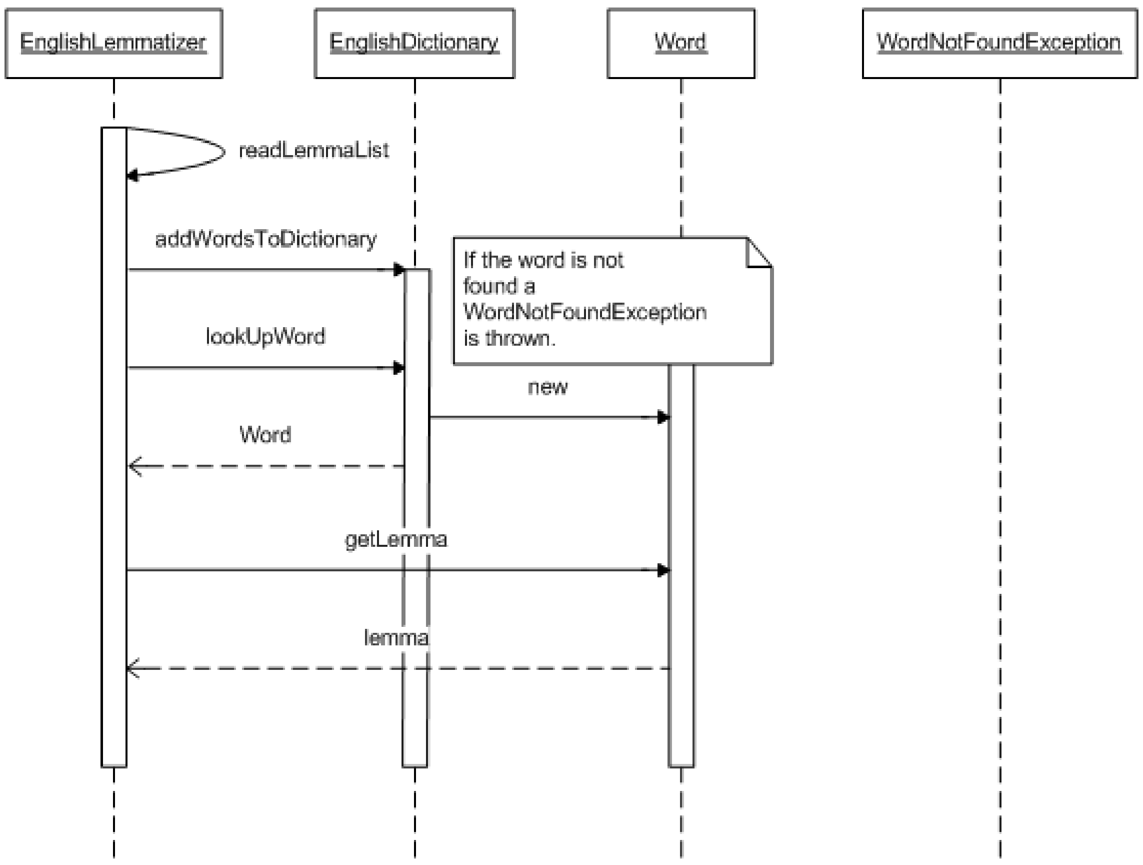
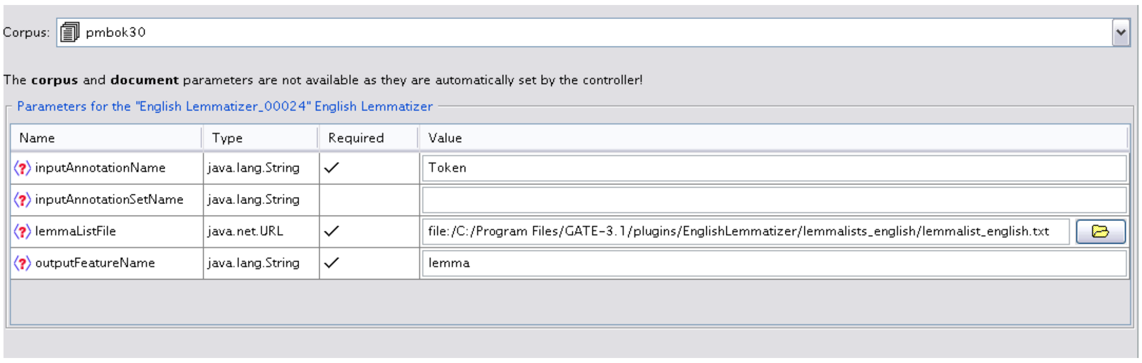


Figure 4.3: Sequence diagram for getting a words lemma

The lemma list is read by the EnglishLemmatizer and each word in the list is added to the EnglishDictionary. The EnglishLemmatizer looks up a certain word, and a Word object is returned. If the word is not found, a WordNotFoundException is thrown. The EnglishLemmatizer then calls the getLemma-method of the Word object and the lemma is returned

The lemmatizer is implemented as a component in GATE. Figure 4.4 show a screenshot of the initialization parameters of the lemmatizer. A shown in the figure, the lemmatizer uses the "Token" (word) annotation resulting from running GATE's built-in tokenizer. The "outputFeatureName", here "lemma" is added as an feature to the "Token" annotation, containing the lemma of the word.


Corpus: pmbok30

The **corpus** and **document** parameters are not available as they are automatically set by the controller!

Parameters for the "English Lemmatizer_00024" English Lemmatizer

Name	Type	Required	Value
inputAnnotationName	java.lang.String	✓	Token
inputAnnotationSetName	java.lang.String		
lemmaListFile	java.net.URL	✓	file:/C:/Program Files/GATE-3.1/plugins/EnglishLemmatizer/lemmalists_english/lemmalist_english.txt
outputFeatureName	java.lang.String	✓	lemma

Figure 4.4: Screenshots of the initialization parameters of the lemmatizer

4.2.2 NounPhrase

The noun phrase module is responsible for finding terms consisting of successive nouns, for example *"project team management"*. This is important when extracting keyphrases used as concept in an ontology. The noun phrase module is dependent on GATE's part-of-speech tagger. After the part-of-speech tagger has annotated each word in each document with its correct part-of-speech, the noun phrase module fetches each word labeled as a noun and searches for successive nouns. The scheme used is *Noun* (*Noun*)*.

The pseudo code for the algorithm can be seen in Figure 4.5.

NounPhrase pseudo code

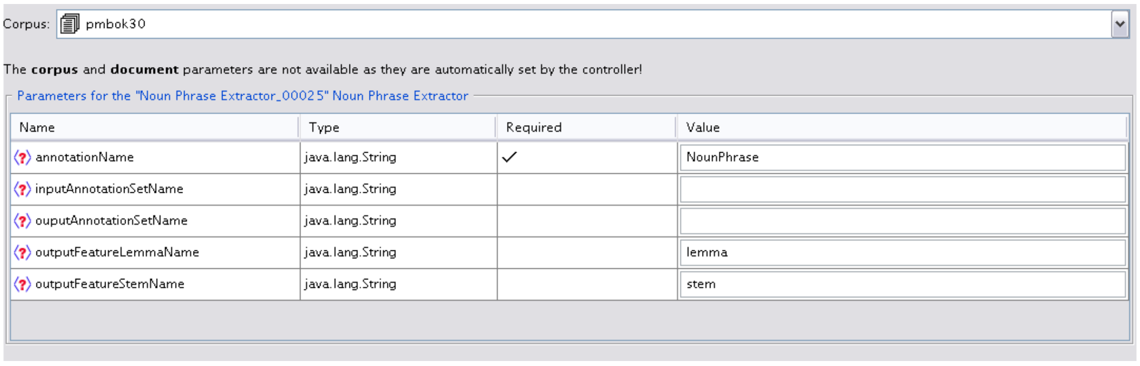
```
for each word in document
    if insideNounPhrase is true
        if word labeled as "Noun"
            add word to nounPhrase
        else
            insideNounPhrase is false
        end if
    else if word labeled as "Noun"
        insideNounPhrase is true
        add word to nounPhrase
    end if
end for
```

Figure 4.5: Pseudo code for the algorithm for finding noun phrases.

The following java classes are contained in the noun phrase module:

- NounPhraseExtractor

The noun phrases found is added as an annotation to each document in the corpus analyzed. The noun phrase extractor is implemented as a component in GATE. Figure 4.6 shows a screenshot of the initialization of the noun phrase extractor. The most notable parameters are "annotationName" and "outputFeatureLemmaName". The "annotationName" is the name of the annotation added to the document. This can be seen in the right corner of Figure 4.7, which shows the resulting annotations after running this component. It should be noted that the annotation viewer is a built-in feature in GATE, connected to a document. The "outputFeatureLemmaName" is the the name of the feature of the annotation, and contains the noun phrase's lemma. This can be seen right above the annotations in Figure 4.7. This is actually a convenience duplication of the feature that the lemmatizer produces.

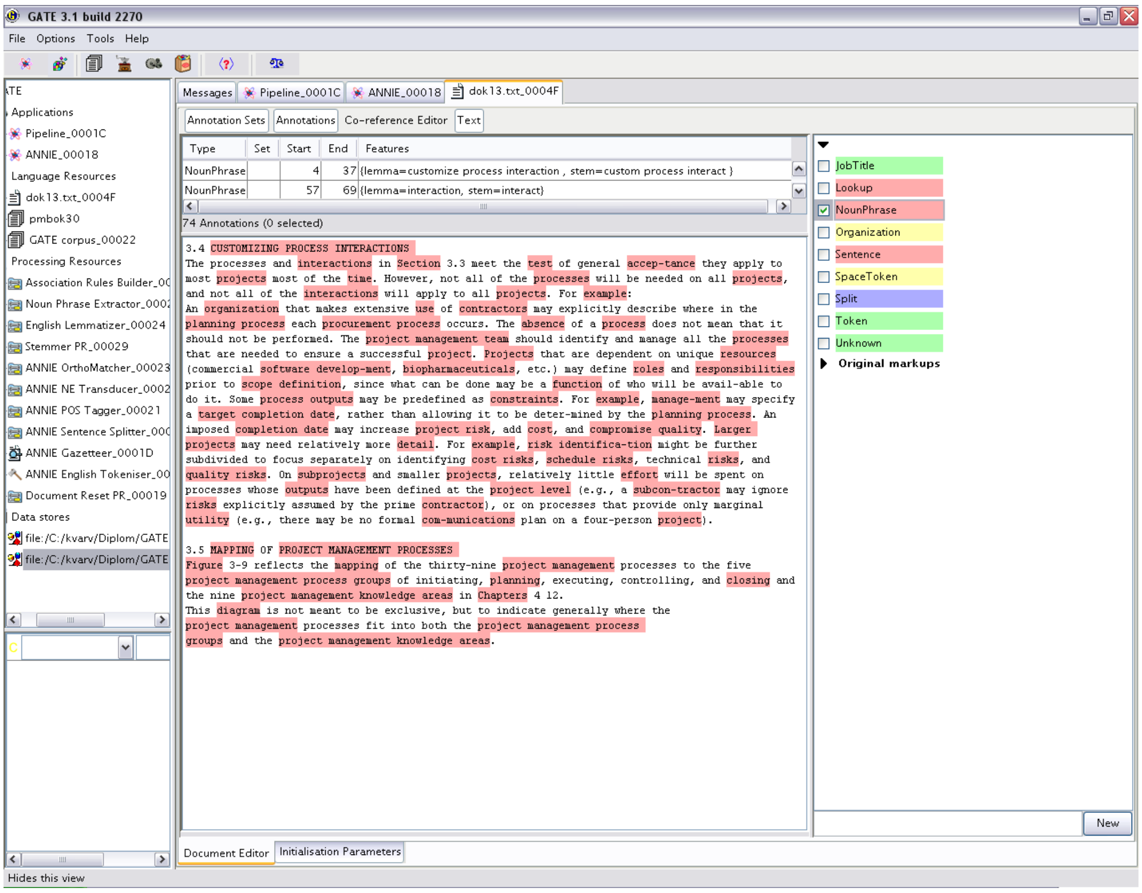

Corpus: pmbok30

The **corpus** and **document** parameters are not available as they are automatically set by the controller!

Parameters for the "Noun Phrase Extractor_00025" Noun Phrase Extractor

Name	Type	Required	Value
annotationName	java.lang.String	✓	NounPhrase
inputAnnotationSetName	java.lang.String		
ouputAnnotationSetName	java.lang.String		
outputFeatureLemmaName	java.lang.String		lemma
outputFeatureStemName	java.lang.String		stem

Figure 4.6: Screenshots of the initialization parameters of the noun phrase extractor



Type	Set	Start	End	Features
NounPhrase		4	37	{lemma=customize process interaction , stem=custom process interact }
NounPhrase		57	69	{lemma=interaction, stem=interact}

74 Annotations (0 selected)

```
3.4 CUSTOMIZING PROCESS INTERACTIONS
The processes and interactions in Section 3.3 meet the test of general accep-tance they apply to
most projects most of the time. However, not all of the processes will be needed on all projects,
and not all of the interactions will apply to all projects. For example:
An organization that makes extensive use of contractors may explicitly describe where in the
planning process each procurement process occurs. The absence of a process does not mean that it
should not be performed. The project management team should identify and manage all the processes
that are needed to ensure a successful project. Projects that are dependent on unique resources
(commercial software develop-ment, biopharmaceuticals, etc.) may define roles and responsibilities
prior to scope definition, since what can be done may be a function of who will be avail-able to
do it. Some process outputs may be predefined as constraints. For example, manage-ment may specify
a target completion date, rather than allowing it to be deter-mined by the planning process. An
imposed completion date may increase project risk, add cost, and compromise quality. Larger
projects may need relatively more detail. For example, risk identifica-tion might be further
subdivided to focus separately on identifying cost risks, schedule risks, technical risks, and
quality risks. On subprojects and smaller projects, relatively little effort will be spent on
processes whose outputs have been defined at the project level (e.g., a subcon-tractor may ignore
risks explicitly assumed by the prime contractor), or on processes that provide only marginal
utility (e.g., there may be no formal com-munications plan on a four-person project).

3.5 MAPPING OF PROJECT MANAGEMENT PROCESSES
Figure 3-9 reflects the mapping of the thirty-nine project management processes to the five
project management process groups of initiating, planning, executing, controlling, and closing and
the nine project management knowledge areas in Chapters 4 12.
This diagram is not meant to be exclusive, but to indicate generally where the
project management processes fit into both the project management process
groups and the project management knowledge areas.
```

Figure 4.7: Screenshot of the resulting annotations from running the noun phrase component in GATE

4.3 Index Module

The index module is responsible for indexing keywords found in the documents. This is done by extracting statistics from each documents. In addition, stop words are removed. These stop words are contained in a user specified file. Keywords are extracted by using the term frequency weighting scheme described in Section 2.3.2 on the noun phrases found by the noun phrase module. The term frequency-inverse document frequency (tf-idf) is not used because it is found by dividing the number of all documents by the number of documents containing the term, and then taking the logarithm of that quotient. The tf-idf would then have led to filtering of common important noun phrases in the corpus, since the ontology learning system is to be used on domain specific text, where the probability of a noun phrase occurring in many documents is high. The stop word list used in this work can be found in Appendix C.

The following java classes are contained in the index module:

- StopWords
- Document
- DocumentStatistics
- Index

4.4 Association Rules Module

The association rules module uses the keywords found in the index for mining association rules. It has two submodules. The generator module generates the rules, whereas the rule searcher module is responsible for searching after specific rules. Figure 4.8 show a sequence diagram of how the association rules module uses the index module for extracting rules.

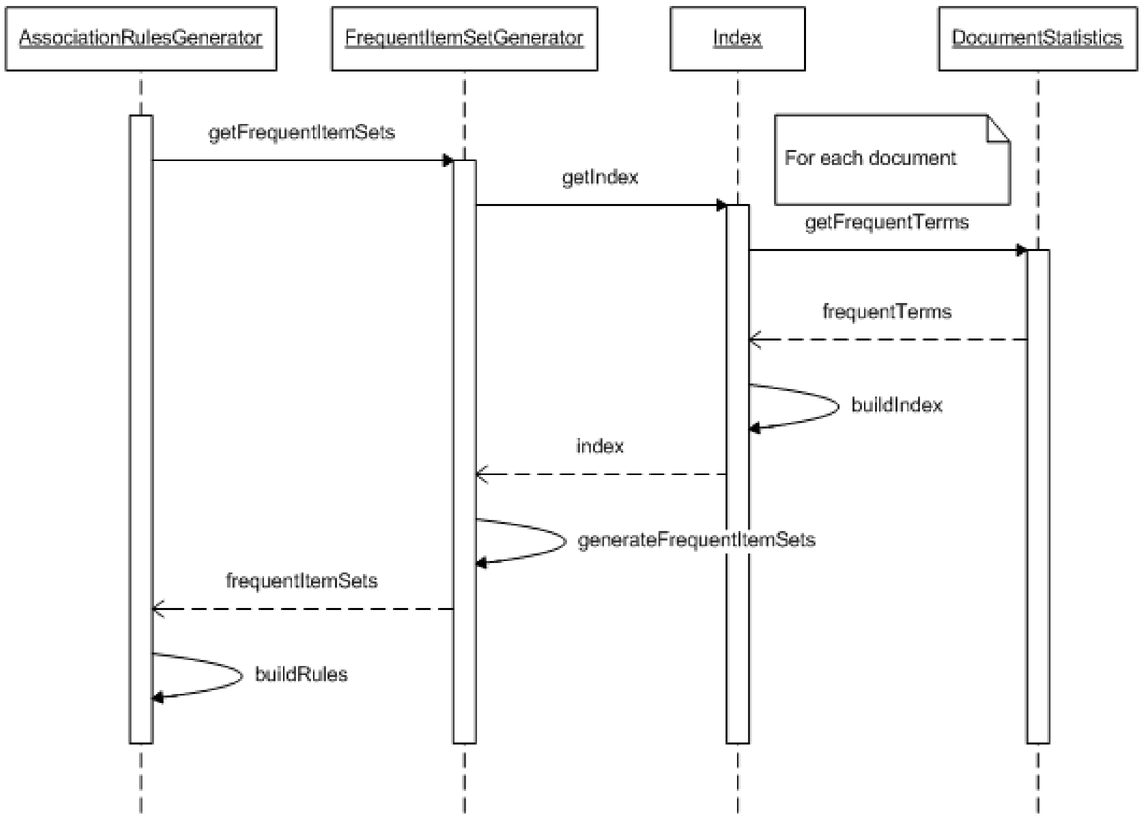


Figure 4.8: Sequence diagram of how the association rules module uses the index module for extracting rules.

4.4.1 Generator

The generator module implements the the association rules algorithm described in Section 3.1. The algorithm implemented uses the foundation of the algorithm found in Agrawal et al. [1993]. However, it is inspired by the java implementation described in Margahny and Mitwaly [2005].

The following java classes are contained in the generator module:

- FrequentItemSetGenerator
- AssociationRulesGenerator

Figure 4.9 and Figure 4.10 shows the core methods from the FrequentItemSetGenerator class.

FrequentItemSetGenerator code snippet

```java
private void findFrequentItemSets() {
    findFirstFrequentItemSets();
    while(doneFindingItemSets == false) {
        generateFrequentItemSets();
    }
}

private void findFirstFrequentItemSets(){
    database = index.getDocumentIndex();
    TreeMap<String, ArrayList<Integer>> frequentItemSets = new TreeMap<String,
        ArrayList<Integer>>();

    Iterator iterator = database.entrySet().iterator();
    while(iterator.hasNext()){
        Entry<String, ArrayList<Integer>> entry = (Entry<String, ArrayList<
            Integer>>) iterator.next();
        String key = entry.getKey();
        ArrayList<Integer> value = entry.getValue();
        if (value.size() >= minimumSupport) frequentItemSets.put(key, value);
    }
    listOfFrequentItemSets.add(frequentItemSets);
    pass++;
}
```

Figure 4.9: Code snippet from FrequentItemSetGenerator class.

FrequentItemSetGenerator code snippet

```java
private void generateFrequentItemSets() {
    int nofMatchingItems = pass - 2;
    TreeMap<String, ArrayList<Integer>> previousFrequentItemSets = (TreeMap<
        String, ArrayList<Integer>>)
                        listOfFrequentItemSets.get(pass-2);

    TreeMap<String, ArrayList<Integer>> currentFrequentItemSets = new TreeMap<
        String, ArrayList<Integer>>();
    ArrayList<String> keys = getKeys(previousFrequentItemSets);

    if (nofMatchingItems == 0) {
        for (int i = 0; i < keys.size(); i++) {
            String currentKey = keys.get(i);
            for (int j= i+1; j < keys.size(); j++){
                String nextKey = keys.get(j);
                String newKey = joinItemSets(currentKey, nextKey);
                ArrayList<Integer> supportList = getSupportList(
                    previousFrequentItemSets, currentKey, nextKey);
                if (supportList.size() >= minimumSupport) currentFrequentItemSets
                    .put(newKey, supportList);
            }
        }
    }
    else {
        for (int i = 0; i < keys.size(); i++) {
            String currentKey = keys.get(i);
            String [] currentKeySplit = itemSetToArray(currentKey);
            String itemToMatch = "";

            for (int x = 0; x < nofMatchingItems; x++) {
                itemToMatch += currentKeySplit[x];
            }

            for (int j= i+1; j < keys.size(); j++){
                String nextKey = keys.get(j);
                String [] nextKeySplit = itemSetToArray(nextKey);
                String matchingItem = "";
                String itemToAdd = nextKeySplit[nofMatchingItems];

                for (int y = 0; y < nofMatchingItems; y++) {
                    matchingItem += nextKeySplit[y];
                }

                if (itemToMatch.equals(matchingItem)) {
                    String newKey = joinItemSets(currentKey, itemToAdd);
                    ArrayList<Integer> supportList = getSupportList(
                        previousFrequentItemSets, currentKey, nextKey);
                    if (supportList.size() >= minimumSupport)
                        currentFrequentItemSets.put(newKey, supportList);
                }
            }
        }
    }
    if (currentFrequentItemSets.size() != 0) listOfFrequentItemSets.add(
        currentFrequentItemSets);
    else doneFindingItemSets = true;
    pass++;
}
```

Figure 4.10: Code snippet from FrequentItemSetGenerator class.

The algorithm runs as follows:

The database TreeMap is loaded with the index (described in Section 4.3). This database contains each keyphrase and an ArrayList with each document that contains the word. The size of the ArrayList is therefore the support of each keyphrase.

The first frequent item set is found by iterating over the database and adding every keyphrase and its associated ArrayList that has support over or equal to a given threshold (minimum support) to a list of frequent item sets. This list of frequent item sets are added to a list that holds the list of frequent item sets generated in each pass.

For every pass the algorithm joins the previous list of frequent item sets by itself such that the *pass-2* first items in the set matches. The ArrayList of the two joined item sets are compared, adding each common document from the two, to form a new ArrayList that now holds the support of the resulting itemset. If the support is greater than or equal to the minimum support the item set is added to the current list of frequent item sets.

The algorithm proceeds until there are no matching *pass-2* item sets.

The advantage of this algorithm compared to the one found in Agrawal et al. [1993], is that the database is loaded in to memory and the algorithm thereby won't have have to read the support of an item set from disc. This increases speed significantly. Another great advantage is that there is no need for generating candidate item sets as stated in both Agrawal et al. [1993] and Margahny and Mitwaly [2005]. This is a consequence of the former advantage, since the support of the newly joined item set is found at once it is generated, and the item set therefore can be removed instantly if the support is below the minimum support.

After generating the frequent item sets the AssociationRulesGenerator class generates the association rules from the frequent item sets. The pseudo code for this algorithm is illustrated in Figure 4.11.

FrequentItemSetGenerator code snippet

```
for (i = 0; i < number of passes; i++)
    get list of frequent item sets from pass i

    for each item set in list of frequent item sets
        for each item in item set
            left hand of rule = item set − item
            right hand of rule = item
            if support of item set divided by support of left hand > minmum
                confidence
                keep rule
            end if
        end for
    end for
end for
```

Figure 4.11: Pseudo code for generating association rules from frequent item sets.

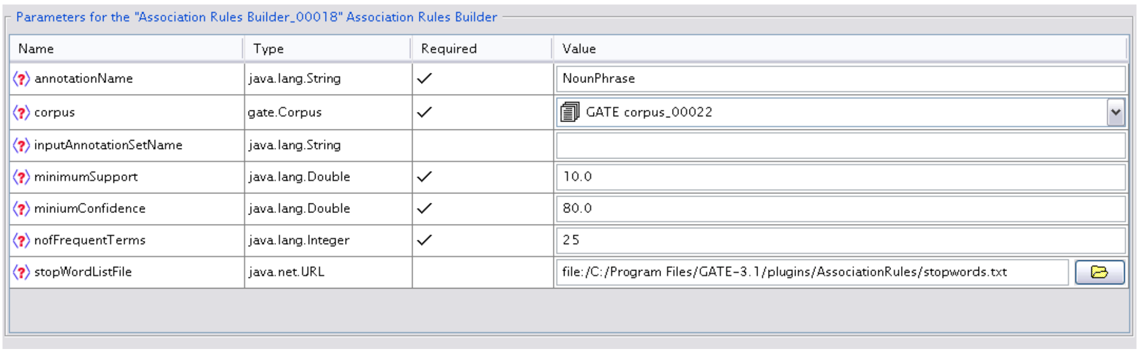
4.4.2 Rule

The rule module is responsible for looking up certain rules generated by the generator module, either by searching on the left hand term, the right hand term or both. It also keeps track of information such as confidence and support for each rule.

The following java classes are contained in the rule searcher module:

- Rule
- RuleSearcher

The association rules module is implemented as a component in GATE. Figure 4.12 shows the initialization of this component. As described earlier in this section, the association rules component uses the index module in the background. The index module can any annotations in a document. This is set with the “annotaionName” parameter. Here it is chosen to index and extract rules based on the “NounPhrase” annotation. Other important parameters for association rules component are “minimumSupport”, “minimumConfidence”, “nofFrequentTerms” and “stopWordFileList”. Minimum support and minimum confidence were described in Section 3.1. “nofFrequentTerms” is the number of terms (here noun phrases) from each document in the document collection that is used to extract association rules. The “stopWordList-File” parameter is the path to file containing the stop words, that is words that will be excluded from the computation of rules.

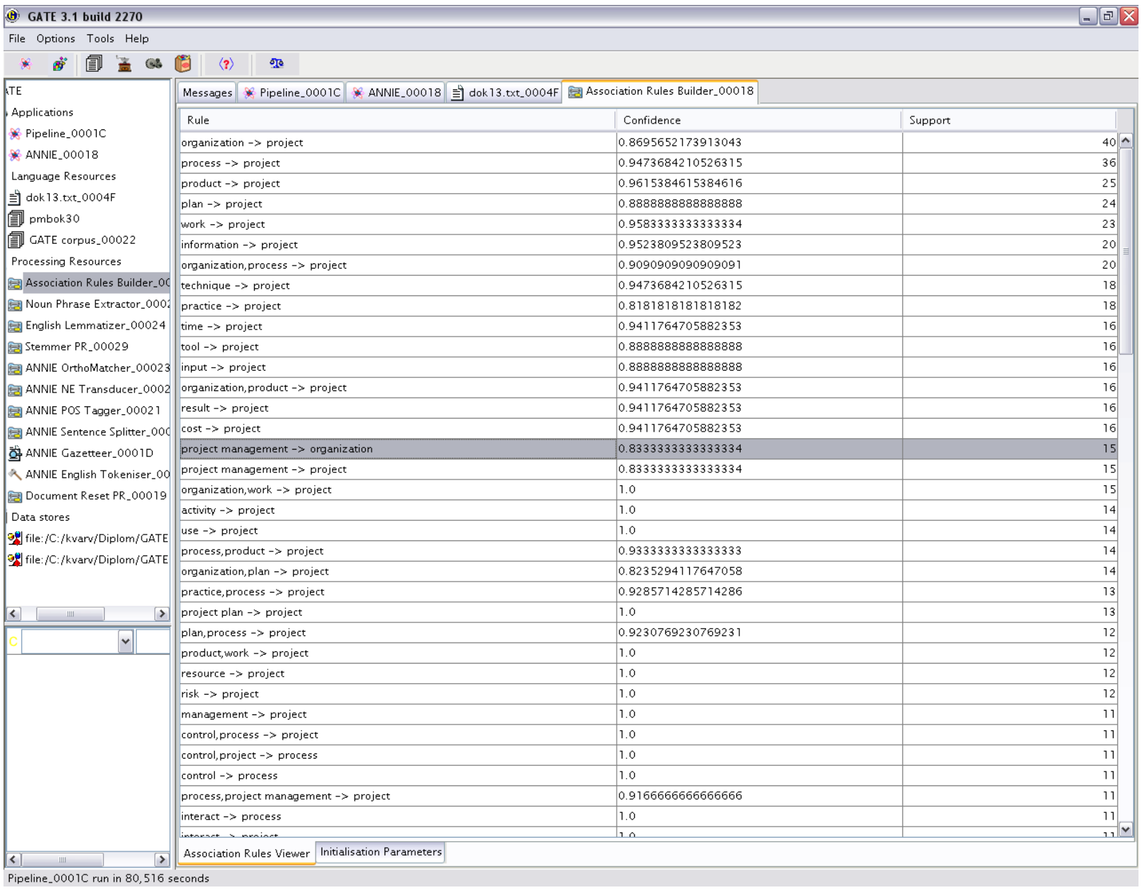


Parameters for the "Association Rules Builder_00018" Association Rules Builder

Name	Type	Required	Value
annotationName	java.lang.String	✓	NounPhrase
corpus	gate.Corpus	✓	GATE corpus_00022
inputAnnotationSetName	java.lang.String		
minimumSupport	java.lang.Double	✓	10.0
miniumConfidence	java.lang.Double	✓	80.0
nofFrequentTerms	java.lang.Integer	✓	25
stopWordListFile	java.net.URL		file:/C:/Program Files/GATE-3.1/plugins/AssociationRules/stopwords.txt

Figure 4.12: Screenshot of the initialization parameters of the association rules component.

Figure 4.13 shows a screenshot of the association rules resulting from running the association rules component. This visual resource was implemented specifically for the association rules component.



Rule	Confidence	Support
organization -> project	0.8695652173913043	40
process -> project	0.9473684210526315	36
product -> project	0.9615384615384616	25
plan -> project	0.8888888888888888	24
work -> project	0.9583333333333334	23
information -> project	0.9523809523809523	20
organization,process -> project	0.9090909090909091	20
technique -> project	0.9473684210526315	18
practice -> project	0.8181818181818182	18
time -> project	0.9411764705882353	16
tool -> project	0.8888888888888888	16
input -> project	0.8888888888888888	16
organization,product -> project	0.9411764705882353	16
result -> project	0.9411764705882353	16
cost -> project	0.9411764705882353	16
project management -> organization	0.8333333333333334	15
project management -> project	0.8333333333333334	15
organization,work -> project	1.0	15
activity -> project	1.0	14
use -> project	1.0	14
process,product -> project	0.9333333333333333	14
organization,plan -> project	0.8235294117647058	14
practice,process -> project	0.9285714285714286	13
project plan -> project	1.0	13
plan,process -> project	0.9230769230769231	12
product,work -> project	1.0	12
resource -> project	1.0	12
risk -> project	1.0	12
management -> project	1.0	11
control,process -> project	1.0	11
control,project -> process	1.0	11
control -> process	1.0	11
process,project management -> project	0.9166666666666666	11
interact -> process	1.0	11
interact -> project	1.0	11

Pipeline_0001C run in 80,516 seconds

Figure 4.13: Screenshot of the resulting association rules from running the association rules component in GATE.

4.5 Ontology Module

For now, the ontology module only consist of one submodule, concept. The idea is that the ontology module should be expanded in the future with other modules that use the associated concepts found by the system, incorporated with the ontology editor found in GATE.

4.5.1 Concept

The concept module is responsible for finding all the associated concepts for each of the concepts extracted by the system. [Haddad et al., 2000] defines the environment of a concept to be the set of concepts related to it. More formally, let S be the set of concepts used in the association rules, $X \Rightarrow Y$ a rule. Then the environment of X

is:

$$\mathrm{Env}(X) = \{Y \in S \mid X \Rightarrow Y\}$$

By applying this for every rule found by the system and adding the right hand side of the rule to the environment of the left hand side, all the associated concepts for each concept will be found. Figure 4.14 shows the pseudo code of the process.

Pseudo code for finding environments

```
for (i = 0; i < number of passes; i++)
    get list of frequent item sets from pass i

    for each item set in list of frequent item sets
        for each item in item set
            left hand of rule = item set - item
            right hand of rule = item
            if support of item set divided by support of left hand > minmum
                confidence
                keep rule
            end if
        end for
    end for
end for
```

Figure 4.14: Pseudo code for finding environments from the rules.

The following java classes are contained in the concept module:

- Concept
- EnvironmentGenerator

Chapter 5

Evaluation Method

This chapter describes a formal objective method of evaluation. The goal of this evaluation is to evaluate the associations found by the association rules algorithm. As the concepts extracted by the system is used as input to the associations rules algorithm, these concepts therefore directly influences the associations found. The following will be evaluated:

- The concepts extracted by the system
- The associations found by the system

To perform this evaluation there have to exist some comparison foundation. The following data foundation is provided:

- Text from the book "A Guide To The Project Management Body Of Knowledge" (PMBOK) [PMI, 2004] and the first six chapters of "Organizational Project Management Maturity Model" [PMI, 2003], split into 76 documents.
- A manually constructed ontology built upon PMBOK.
- A semi-automatically constructed ontology built upon PMBOK.
- Results from cosine similarity calculations between the concepts in the manually constructed ontology.

The system will be tested with PMBOK as input. Table 5.1 shows which of the provided data the concepts and the associations will be compared with.

	Concepts	Associations
Manually constructed ontology	X	
Semi-automatically constructed ontology	X	
Cosine similarity results		X

Table 5.1: Overview of the evaluation

5.1 Concepts

For a best possible evaluation of the concepts found by the system, they ought to be compared to some “gold standard” for the given domain or approved by a domain expert. However, none of these were available. The only available material for evaluation is the work done by Grimnes [2006] which was based on upon the same domain and the same input text. This will be described in further detail in the next section. The evaluation of the concepts found by the system is important because these concepts are the input of the association rules algorithm, and the associations found will be a direct consequence of the concepts. Knowledge about the overlap between the concepts found by the system and the concepts in the manually constructed ontology is therefore important for the evaluation of the associations found. This will be described in further detail in Section 6.2. As concept extraction was not an important part of this project, the solution is rather simplified with a lot of improvement potentiality. The main focus was the association rules.

5.1.1 Evaluation Strategy

The concepts found by the system will be compared with the concepts found in both the manually constructed ontology and the semi-automatic constructed ontology found in Grimnes [2006], as described in Section 2.2. The concepts in the ontologies are extracted from the same set of documents, described in the beginning of the chapter, used as input for the system the system. The manually constructed ontology contains 142 concepts approved by a domain expert. The semi-automatically constructed ontology is used for comparing the method for extracting concept used in the system and the method used in Grimnes [2006]. The ontology contains 106 concepts. Since the system uses the lemma of the concepts, the concepts found in the two ontologies will be lemmatized before the comparison is performed. The two ontologies can be found in Appendix C

The overlap will be tested in the following way:

1. Check for perfect match between the concepts.

Each concept found by the system will be evaluated for equality between the concepts found in the ontologies.

2. If not a perfect match, check for abstraction match.

   The concept “project management team” has two abstraction levels, “management team” on the first level and “team” on the second level. If a given concept contained in the ontologies is not found among the concepts found by the system, the given concept will be evaluated whether its abstraction is found by the system. For example, if the ontology contains the concept “project team” and this concept is not among the concepts found by the system, the concept will be evaluated whether “team” is found by the system. This yields a partial match of the concept.

5.2 Associations

The evaluation of the associations found by the system is the most important part of the evaluation. Extraction of associations between concepts has been the main objective for this project. The association rules algorithm outputs a set of rules that indicates an association between two or more concepts. The associations found ought to be evaluated by a domain expert for a best possible evaluation. As described in the beginning of this chapter, a domain expert was not available. Another approach for finding related concepts is the cosine similarity model described in Section 2.3.3. Solskinnsbakk [2006] performed a cosine similarity test between the concepts in the manually constructed ontology found in Grimnes [2006]. The results from these tests will be used for evaluation of the associations found by the system. However, there is no knowledge whether the relations found by cosine similarity test are good. Also, since similarity were calculated between the concepts in the manually constructed ontology, the overlap between them and the concepts found by the system could bias the results. The association found by the system and the results from the cosine similarity test will therefore be evaluated by 4 master students, in the lack of a domain expert. Since project management is a rather general domain and the master students have some experience with project management from different projects, this should be good enough to see trends in the approach. This will be described in further detail in the next section. The cosine similarity results can be found in Appendix C.

5.2.1 Evaluation Strategy

As described in Section 6.2, the cosine similarity results from Solskinnsbakk [2006] will be used for evaluation of the relations. The associated concepts found by the system will be compared the top 50 related concepts from the cosine similarity results. Five of the concepts found by the system will be evaluated. The randomly chosen concepts are “control”, “scope”, “cost”, “risk” and “project plan”.

The overlap will be tested in the same way as for the concept evaluation:

1. Check for perfect match between the related concepts.

   For each of chosen concepts, all the associated concepts will be evaluated for equality of the related concepts found by the cosine similarity method.

2. If not a perfect match, check for abstraction match.

   If the given concept is not among the related concepts found by the system, the given concept will be evaluated whether its abstraction is found, in the same manner as the concept evaluation.

Three important questions arise:

1. Are the associations found by the system, but not found by the cosine similarity good or bad?
2. Are the associations found by the cosine similarity, but not found by the system good or bad?
3. Are the associations found by both methods good?

Based on these questions the results from the overlap evaluation will be split into three groups:

- **Group 1**: Related concepts found only by the system, not by cosine similarity.
- **Group 2**: Related concepts found only by cosine similarity, not by the association rules.
- **Group 3**: Related concepts found by both.

Figure 5.1 illustrates these three groups, resulting from the overlap between the related concepts found by the system and the cosine similarity method.

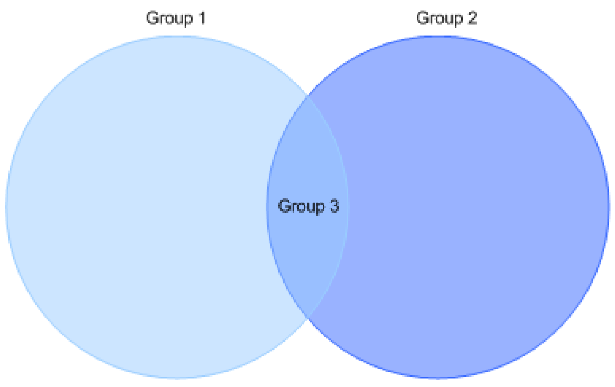


Figure 5.1: A venn diagram of the groups resulting from the overlap between the related concepts found by the system and the cosine similarity method.

As the described earlier the associations found by the system and the related concepts found by the cosine similarity test will be evaluated by 4 master students at IDI. For each of the 5 chosen concepts, the associated concepts found will be evaluated for relatedness by the students. The possible scores are illustrated in Table 5.2.

2	Very related
1	Related
0	Not related

Table 5.2: Possible evaluation scores of related concepts

By comparing the results according to each group described above, the evaluation can give an indication of the better method of association rules and cosine similarity. It would at least indicate how well suited the association rules are as a method for finding associations between concepts.

The results from the evaluation by the 4 master students can be found in Appendix C.

5.3 GATE

One of the objectives of this project was to evaluate how well suited GATE is as a framework for ontology learning. A common framework for ontology learning is important because:

- Ontology learning utilizes several different processing components.

- Different situations requires different processing components.
- An analysis usually consists of several processing components nested together.

The evaluation of GATE as a suited framework will consist of a subjective opinion from the experience of using GATE during this work. The following aspects will be emphasized:

- How well is GATE suited for nesting processing components?
- How easy is it to create new, customized processing components?
- How can the results from created processing components be presented?

Chapter 6

Evaluation Results

This chapter presents and discusses the results from the described method of evaluation for the concepts and the associations found by the system.

6.1 Concepts Evaluation Results

This section presents the overlap between the concepts found by the system and the manually constructed ontology, and the overlap of semi-automatically constructed ontology.

The results from running the tests described in Section 5.1.1, evaluating against the manually constructed ontology are shown in Figure 6.1.

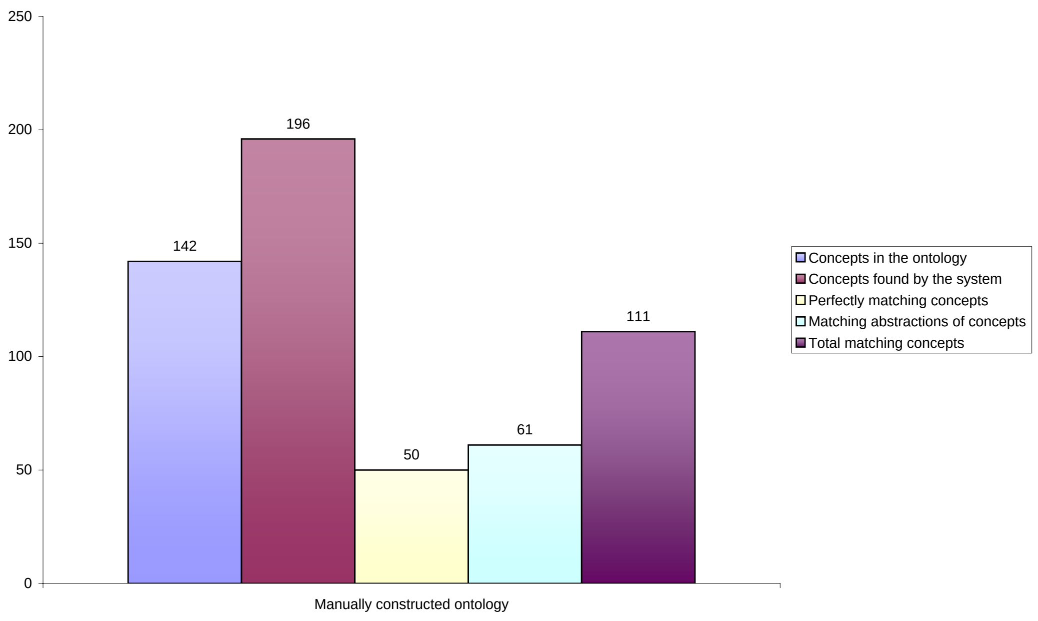


Figure 6.1: Chart of the concepts found by the system, the concepts in the manual constructed ontology and the overlap between them.

As the figure shows, a set of 196 concepts were extracted by the system by using term frequency filtering and running the association rules algorithm. This means that every concept extracted have an association to one or more concepts in the set. 35.2% of the concepts found matched perfectly with the manually constructed ontology. Evaluating against the abstraction of the concepts in the ontology resulted in a match of 43.0%. This means a total match of 78.2%.

A rather large portion of the concepts in the ontology were matched by abstraction. The reason for this is that the ontology contains a large set of specialized concepts, whereas the system extracts more generalized concepts. This is an expected result by using a term's document frequency for concept extraction. A more generalized concept has a higher probability for occurring often in a document than a specialized one.

The results from running the overlap tests described in Section 5.1.1, evaluating against the semi-automatically constructed ontology are shown in Figure 6.2.

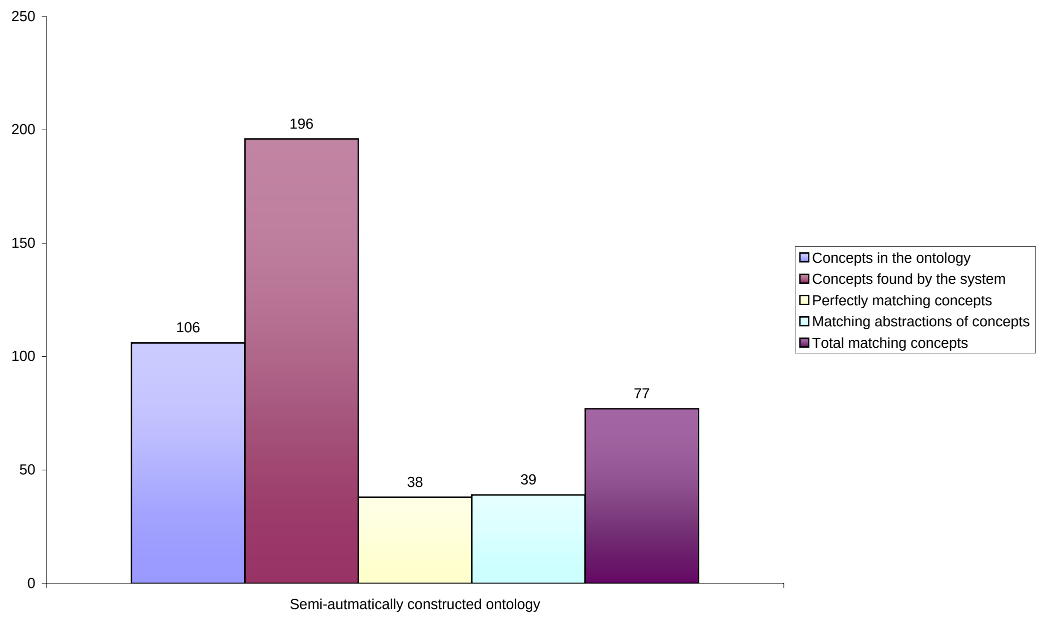


Figure 6.2: Chart of the concepts found by the system, the concepts in the semi-automatically constructed ontology and the overlap between them.

As the figure shows, the concept extraction system used in Grimnes [2006] extracted a set of concepts where 106 of these were approved by a domain expert. 35.8% of these were matched perfectly by the system. Evaluating against the abstraction of the concepts in the ontology resulted in a match of 36.8%. This gives a total match of 72.6%.

Since the concepts were extracted from the same set of documents, it would have been expected that the overlap was higher. Especially since the system extracted 90 more concepts. This could be explained by the fact that the concepts in semi-automatically ontology were extracted by using the tf-idf weighting scheme described in Section 2.3.2, whereas the system uses the term frequency (tf) weighting scheme. Tf-idf extracts more specialized concepts as it gives a low weight to concepts occurring in many documents. The more specialized a concept is, the less the probability that it occurs in many documents. Tf, on the other hand, gives a high weight if a concept occurs often in a document, independently of its distribution throughout the document collection. Using the tf weighting scheme therefore tends to extract more general concepts, as the probability for a general concept occurring often in a document is higher the more general it is. This should make evidence that using only tf for concept extraction is not a satisfactorily method, as it tends to filter out the more specialized concepts.

6.2 Associations Evaluation Results

This section presents the results from the evaluation of the associations found by the system. The evaluation has taken as basis the association rules algorithm with a minimum support of 5%, minimum confidence 75% and the 45 most frequent noun phrases from each document. As described in Section 6.2, the results from the overlap between the associated concepts found by the system and the ones found by cosine similarity will be split into three groups:

- **Group 1**: Related concepts found only by the system, not by cosine similarity.
- **Group 2**: Related concepts found only by cosine similarity, not by the association rules.
- **Group 3**: Related concepts found by both.

6.2.1 Overlap between cosine similarity

Figure6.3 shows the overlap between the related concepts found by the system and the related concepts found by cosine similarity for each of the chosen concepts.

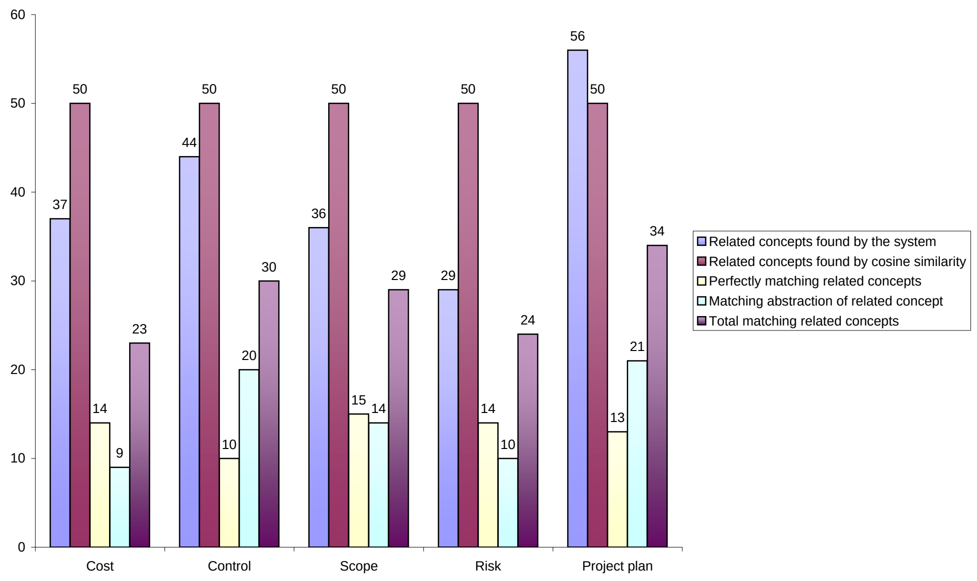


Figure 6.3: Chart of the associations found by the system compared to the cosine similarity results for each of the chosen concepts .

For four of the five chosen concepts, the set of related concepts found by the system was a bit smaller than the set chosen from the cosine similarity results. In addition 31 of the concepts found in the manually constructed ontology, used in the cosine similarity calculations, was not extracted by the system. This constitutes uncertainty factors with the the test. Still there was 20.0%-30.0% perfectly matching related concepts. Taking abstraction matches into account gives a total of 46.0%-68.0% overlap between the related concepts found by the system and the related concepts found by cosine similarity. This seems promising taken into account the uncertainty factors. The next section will however present the subjective evaluation of the 4 master students for both the related concepts found by the system and the related concepts found by cosine similarity. The results from the overlap tests will be split into groups as described in Section 6.2.

6.2.2 Results for each group

As described in Section 6.2 the results from the overlap tests will be split into three groups. Group 1 consisting of the related concepts found only by the system, group 2 consisting of the related concepts found only by cosine similarity and group 3 consisting of related concepts found by both the system and cosine similarity. By looking at each group in terms of the master student's evaluations, this could result in very interesting findings. Only unique related concepts are included for each chosen concept in the groups, meaning that only perfectly matching related concepts are taken into consideration for group 3. For Group 1 this means that abstraction matches are not included. For group 2 it means that if a related concept was matched by abstraction by the system, it is not included.

Group 1 - Related concepts found only by the system

Tables 6.1-6.5 shows the related concepts found only by the system and the mean score of the evaluation of the four master students for every related concept. Figure 6.4 shows how many of the concepts found were evaluated to “not related”, “related” and “very related” for each of the chosen concepts.

Found only by the system for concept ”cost”

Related concept	**Mean score**
project management team	1
management team	1
organization	1
product	2
information	1
tool	1
project team	1
application area	1
risk analysis	1
result	1
risk	1
resource	1
consequence	1
estimate	1
phase	0
probability	1
action	1
analysis	1
seller	2

Table 6.1: Related concepts found only by the system for the concept ”cost”.

Found only by the system for concept ”control”

Related concept	**Mean score**
capability	0
domain	0
improvement	1
organization	1
process improvement	1
program	1
opm	0
practice	0
tool	0
portfolio management	1
management process group	1
result	1
concept	0
group	1
information	1
management process	1
process group	0
standard	1
portfolio	0
resource	1
activity	0
application	1
product	1
action	1
knowledge	0
operation	1
organizational project	1
outcome	1
strategy	1
project performance	1
initiate	0
management team	1

Table 6.2: Related concepts found only by the system for the concept ”control”.

Found only by the system for concept ”scope”

Related concept	**Mean score**
concept	1
operation	1
organization	1
tool	1
product	1
document	1
technique	1
result	1
information	1
project performance	1
resource	1
domain	1
opm	0
group	0
wbs	1
project scope	2
practice	1
risk	1
application area	1
management process	1

Table 6.3: Related concepts found only by the system for the concept ”scope”.

Found only by the system for concept ”risk”

Related concept	**Mean score**
organization	1
result	1
impact	1
tool	1
information	0
project team	1
management team	0
project risk	2
resource	1
plan	1

Table 6.4: Related concepts found only by the system for the concept ”risk”.

Found only by the system for concept ”project plan”

Related concept	**Mean score**
action	0
change	1
change control	1
change request	0
corrective action	1
performance measurement	1
performance report	0
change control system	0
control system	1
tool	1
information	1
organization	1
project management team	1
management team	1
product	1
project performance	2
result	1
variance	1
plan execution	2
activity	1
cost	1
document	1
project team	1
phase	2
analysis	1
application area	1
contract	1
control	1
risk	1
technique	1
project scope	2
relationship	0
work result	1
risk analysis	1
budget	2
group	1
seller	1
resource	1
contingency plan	1
product description	1
response	0

Table 6.5: Related concepts found only by the system for the concept ”project plan”.

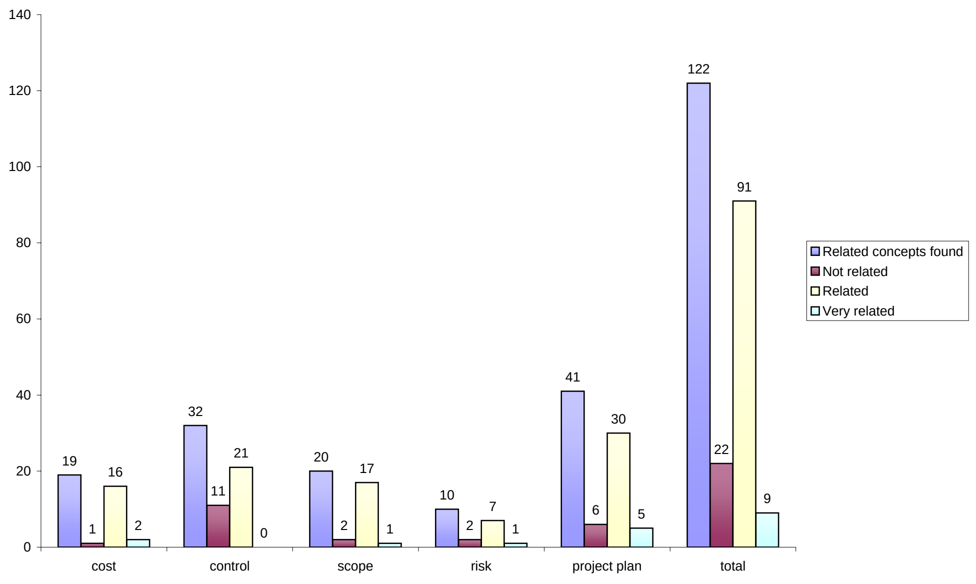


Figure 6.4: Chart of the evaluated related concepts found by the system for each of the chosen concepts.

As Figure 6.4 shows, there is a large portion of the related concepts found that were evaluated to “related” by the 4 master students. 74.6% of the total related concepts found were evaluated to “related”, whereas 18.0% were evaluated to “not related” and only 7.4% were evaluated to “very related”. This will be discussed later.

Group 2 - Related concepts found only by cosine similarity

Tables 6.6-6.10 shows the related concepts found only by the cosine similarity and the mean score of the evaluation of the 4 master students for every related concept. Figure 6.5 shows how many of the concepts found were evaluated to “not related”, “related” and “very related” for each of the chosen concepts.

Found only by cosine similarity for concept ”cost”

Related concept	**Mean score**
cost management	2
cost baseline	2
actual cost	2
schedule	1
project schedule	1
earn value	1
staff	1
project staff	1
milestone	0
plan value	1
stakeholder	2
project deliverable	1
ev	0
earn value management	1
management	1
scope definition	0
scope management	1
customer	1
sponsor	1
project management information system	1
constraint	1
project manager	1
project plan development	1
procurement management	0
project plan execution	0
quality management	1
work breakdown structure	1

Table 6.6: Related concepts found only by cosine similarity for the concept ”cost”.

Found only by cosine similarity for concept ”control”

Related concept	**Mean score**
integrate change control	2
change control	2
schedule control	2
cost control	2
quality control	2
plan value	1
quality	1
scope change control	2
initiation	1
project manager	2
tool and technique	0
schedule	2
performance	0
project management information system	1
project plan execution	1
change control system	2
project plan development	1
project schedule	1
milestone	0
customer	0

Table 6.7: Related concepts found only by cosine similarity for the concept ”control”.

Found only by cosine similarity for concept ”scope”

Related concept	**Mean score**
scope statement	2
scope definition	2
project deliverable	1
project justification	1
project objective	2
plan value	1
milestone	1
performance	1
project plan development	1
customer	1
project plan execution	1
project management information system	1
sponsor	1
initiation	1
scope verification	2
staff	1
project charter	0
quality	1
communication	0
project report	1
work breakdown structure	1

Table 6.8: Related concepts found only by cosine similarity for the concept ”scope”.

Found only by cosine similarity for concept ”risk”

Related concept	**Mean score**
risk identification	2
risk category	2
risk monitor	2
risk control	2
external stakeholder	1
stakeholder risk tolerance	2
sponsor	0
milestone	0
plan value	1
project report	1
project life cycle	1
stakeholder	1
management	1
customer	1
control	1
scope management	1
quality	1
cost management	1
performance	1
project plan execution	1
project manager	1
communication	1
project plan development	1
procurement management	0
project management information system	1
staff	1

Table 6.9: Related concepts found only by cosine similarity for the concept ”risk”.

Found only by cosine similarity for concept "project plan"

Related concept	**Mean score**
project plan development	2
plan value	2
milestone	2
project objective	2
project deliverable	1
project report	1
project charter	1
scope statement	1
project role	1
staff	1
communication	1
constraint	1
customer	1
assumption	0
procurement	0
initiation	1

Table 6.10: Related concepts found only by cosine similarity for the concept "project plan".

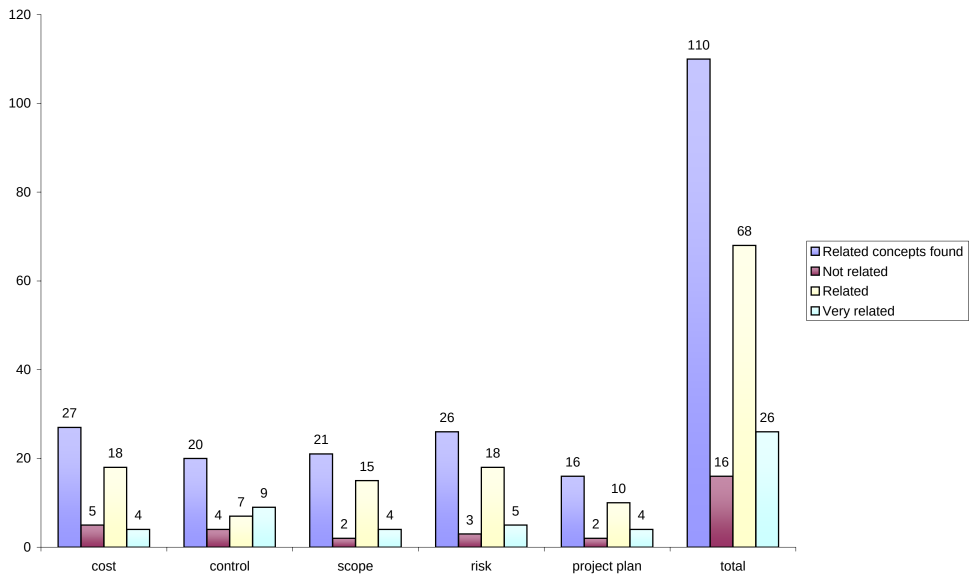


Figure 6.5: Chart of the evaluated related concepts found by cosine similarity for each of the chosen concepts.

Figure 6.5 shows that a larger portion of the related concepts found only by cosine similarity were evaluated to “very related” than the ones only found by the system. 23.6% of total concepts found is in the “very related” category. However, 61.8% were evaluated to “related”, in contrast to 74.6% by the system. It therefore seems that the related concepts found only by the system tends to be more generally related. This fact could be explained by the nature of the association rules algorithm. For an association rule to be generated, the concepts taking part in the rule have to occur together in several documents. Statistically the more general the concept is, the more often it appears in a document. The association rules algorithm therefore finds more generally related concepts. This could also be explained by the lack of overlap between the concept extracted and the concepts in the manually constructed ontology, where only 35.2% of the concepts were matched perfectly. However, there was an abstraction match of 43.0%. This shows that already at the starting point the system had extracted a large set of general concepts in which the association rules found are a direct consequence of. This could also be the reason behind the larger set of related concpets found by cosine similarity evaluated to “very related” than the ones found by the system. They seem to be more precise and therefore are easier to evaluated. In addition there are some fewer related concepts found by cosine similarty evaluated to “not related”. The reason is most likely the fact that the concepts extracted by the system contains some noise. This supports the theory that the concept extraction in the system is not satisfactorily.

Group 3 - Related concepts found by both

Tables 6.11-6.15 shows the related concepts found by both the system and cosine similarity, and the mean score of the evaluation of the 4 master students for every related concept. Figure 6.6 shows how many of the concepts found were evaluated to “not related”, “related” and “very related” for each of the chosen concepts.

Found by both for concept ”cost”

Related concept	**Mean score**
activity	1
assumption	0
control	1
cost estimate	2
performance	1
process	1
project	2
project management	1
project objective	1
project plan	1
quality	1
scope	1
scope statement	1

Table 6.11: Related concepts found by both the system and cosine similarity for the concept cost.

Found by both for concept ”control”

Related concept	**Mean score**
change	1
cost	1
management	2
plan process	1
process	1
project	1
project management	2
project plan	2
scope	1

Table 6.12: Related concepts found by both the system and cosine similarity for the concept control.

Found by both for concept ”scope”

Related concept	**Mean score**
activity	1
assumption	1
change	1
constraint	1
control	1
cost	1
management	1
process	1
project	2
project management	1
project manager	1
project plan	2
schedule	1
stakeholder	0

Table 6.13: Related concepts found by both the system and cosine similarity for the concept scope.

Found by both for concept ”risk”

Related concept	**Mean score**
activity	1
assumption	1
cost	1
process	1
project	2
project management	1
project objective	1
project plan	1
risk analysis	2
risk management	2
risk response	2
schedule	1
scope	1

Table 6.14: Related concepts found by both the system and cosine similarity for the concept risk.

Found by both for concept ”project plan”

Related concept	**Mean score**
management	2
performance	1
plan process	2
process	1
project	2
project management	2
project manager	1
project phase	2
project plan execution	2
schedule	2
scope	1
stakeholder	1

Table 6.15: Related concepts found by both the system and cosine similarity for the concept project plan.

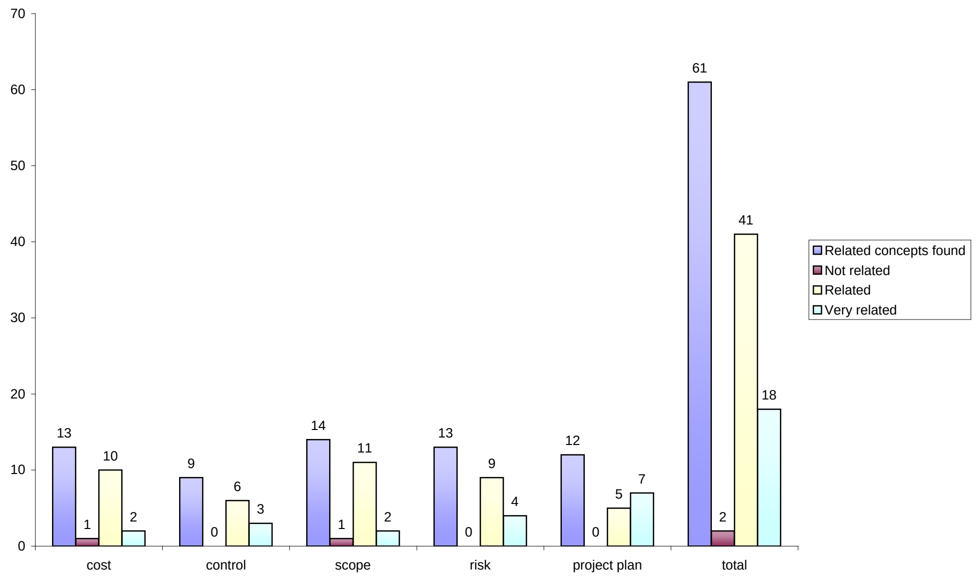


Figure 6.6: Chart of the evaluated related concepts found by both the system and cosine similarity for each of the chosen concepts.

Figure 6.6 shows that only 3.3% of the related concepts found by both cosine similarity and the system were considered "not related". This a very promising result. 67.2% were evaluated to "related" and 29.5% were considered "very related". This shows that the related concepts found by both methods tends to be good.

Group summary

Table 6.16 summarizes the results for each group and shows the percentage of the associated concepts found that were evaluated to "not related", "related" and "very related".

	Not related	Related	Very related
Group 1 - Related concepts found only by the system	18,0 %	74,6 %	7,4 %
Group 2 - Related concepts found only by cosine similarity	14,5 %	61,8 %	23,6 %
Group 3 - Related concepts found by both	3,3 %	67,2 %	29,5 %

Table 6.16: Group summary

Most of the numbers in Table 6.16 have already been discussed. However, the relatively large difference in "very related" concepts between the related concepts found by cosine similarity and the system has more aspects that needs to be discussed. As already stated, these concepts found by cosine similarity seems to more specialized. Looking at Tables 6.6-6.10 shows that not only are they more specialized, but these related concepts seems to include the "parent" concept itself. Take the concept "control" for example. 6 of the 20 related concepts found only by cosine similarity for this concept turns out to be a specialization of "control". Obviously, these 6 will and have been evaluated to very related to the concept "control". The reason for this can be explained by the nature of cosine similarity. It compares the vectors, the contextual categorization of the word. The vector contains information about the context in which the word appears throughout the document collection. Words are similar in the extent that they co-occur with the same words. In another way, two words are found similar if they both co-occur with a third word [Manning and Shutze, 2002]. The association rules algorithm on the other hand, searches for co-occurrence of words in a document in a document collection. It uses statistical measures of how many documents these words co-occur in. The more general the words are, the higher is the probability that words co-occur in many documents. As a consequence it finds associations that seems to be more general. This could explain the difference in the number of concepts evaluated to "very related". Therefore, it seems to be a trend that top-ranked related concepts found by the cosine similarity method includes the concept itself in some way. It can be discussed whether this is good or bad. Some might say that these related concepts are uninteresting and brings little new knowledge to the domain, whereas the association rules algorithm seems to find more general and perhaps a larger set of interesting associations by the cost of some noise. However, a stand will not be taken in this question.

The main findings from this evaluation are as follows:

- The method for concept extraction used in the system is not satisfactory and has great improvement potential. Concept extraction was, however, not an objective of this work. These improvements will be elaborated further in Chapter 8.
- The cosine similarity method seems to find more specialized related concepts.
- The related concepts found by the cosine similarity method tends to include variants of the parent concept.
- The association rules algorithm tends to find more general related concepts.
- The related concepts found by both methods tends to be of good quality.

6.3 GATE

This section gives a subjective opinion of how well suited GATE is as a framework for ontology learning.

GATE as a framework has several advantages:

- Graphical user interface.

  Annotations resulting from processing documents can be directly viewed. Performing tasks is easy.
- Processing components can be nested.

  By creating a processing pipeline, different processing components can be added and run together. The order of the processing components can easily be switched.
- Processed documents can be stored.

  Processed documents can be stored and loaded at any time, with annotations. It is therefore only necessary to process a document once. This is a great feature when a processing component is dependent on annotations resulting from running another component.

- Large set of built-in processing components.
- Easy to create new processing components.

  Using the GATE API makes it easy to create new customized processing components.
- Ontology Editor.

  GATE has an ontology editor where ontologies can be viewed and edited.

The ease of making new components is one of GATE's strengths. Both processing components and visual components can be added by ease with GATE's API. In this work all the components implemented was first created as stand-alone applications outside GATE by using its API. These were included into GATE by adding a small portion of code, which was described very well in the user guide. New visual components can as easily be added and connected to a processing component. Such a visual component is shown i Figure 4.13 in Section 4.4.

However, there is one set back with using GATE. When processing a stored document collection, the load time for each document is a somewhat high. The mean load time for the document collection used as input for the system was 0.88 seconds. The document collection included 76 documents. The total run time for the association rules algorithm, including document loading was 6 minutes and 57 seconds. When analyzing large document collections with thousands of documents, the load time can be a problem. This is in many situations compensated for by the ability to store and load annotated documents.

The bottom line is that GATE is an excellent and highly scalable framework for natural language processing, which is well suited for ontology learning. The combination of the ontology editor and ease of making new customized processing components should make it possible to manually add extracted concepts and associations directly to an ontology within the graphical user interface. It should be stressed that GATE has been used with a small document collection. How well it is suited for extremely large document collections remains unanswered and ought to be evaluated further.

6.4 Evaluation Summary

As pointed out in this chapter, the evaluation preformed has its limitations and uncertainty factors:

- The evaluation of related concepts found are the subjective opinion of four persons. The evaluation should have been performed by a domain experts.

- Several of the concepts used in the cosine similarity test was never extracted by the system.
- The system was tested with a small document collection.

These factors evidently do not make the evaluation adequate to draw any conclusions. Still, the results shows, taking the uncertainty factors into account, that the use of association rules to find associations in text has potential. The approach should be evaluated further by using a better concept extractor and tested with a larger document collection, and the resulting associations should be evaluated by a domain expert. This could lead to more clear conclusions.

The experience of using GATE as a framework for ontology learning has shown that it is highly scalable and customizable. It has several built-in components that makes it useful for ontology learning. However, the framework ought to be tested with a large document collection.

Chapter 7

Conclusion

This master thesis has examined the use of association rules for suggesting associations from text. This has been part of the ongoing research in the field of ontology learning. The objectives of this work were:

1. Implement an association rules algorithm for suggesting associations from text.

   The association rules algorithm and several other components in which the association rules algorithm is built upon, such as a lemmatizer, a noun phrase extractor, a index and a concept extractor, has been implemented and integrated in GATE, as a framework for ontology learning.

2. Evaluate the results against existing ontology.

   The evaluation has indicated that the associations found by the system are of good quality. The main findings were that the compared method of cosine similarity seems to find more specialized related concepts. However, they tend to include variants of the parent concept itself. The association rules algorithm on the other hand, tends to find more general related concepts. It is, however, believed that improving the concept extraction will yield even better results. Even so, the associations found by the system should be evaluated by a domain expert.

3. Evaluate the usability of GATE as an overall framework for ontology learning.

   The GATE framework has been found to be suited for ontology learning. It has several aspects that makes it justifiable to use it as a framework for ontology learning. Still, the framework should be tested with a larger document collection.

With a proper improvement of the concept extraction, and a proper evaluation by a domain expert of the associations found by the system, association rules could be a step toward the goal of finding relations between concepts in an ontology learning environment.

Chapter 8

Further Work

Although the evaluation indicates that the association rules algorithm has potential, there are possibilities for improvements in the application of this algorithm in ontology learning. These improvements does not lie in the algorithm itself, but in the extraction of concepts given as input to the algorithm and how the association rules generated can be used further.

8.1 Concept Extraction

As concept extraction was not emphasized in the work, this have some improvement potential. Term frequency was used to weight the noun phrases in each document. For each of the documents the k top-ranked phrases were extracted. This could lead to some noise. The tf-idf weighting score would filter out most of this noise. However, this weighting scheme would also potentially filter out prominent important noun phrases for a domain specific document collection.

There is a noticeable difference between language used in specialist communication and the language used to to communicate with a more general audience. Gillam and Ahmad [2005] explored a method that exploits this difference, which contrasts term frequencies in a domain specific document collection with term frequencies from a general language document collection. This resulted in a weirdness frequency defined as:

$weirdness = \frac{N_{GL} f_{SL}}{(1+f_{GL}) N_{SL}}$, where

f_{SL} = the frequency of a term in the domain specific document collection.
f_{GL} = the frequency of the term in a general language document collection.
N_{SL} = the token counts of the domain specific document collection.
N_{GL} = the token counts of the general language document collection.

This could be applied to the candidate phrases found by the system, which would probably lead to a set of more domain specific concepts. As the association rules algorithm is dependent of co-occurrence of phrases in a document, this method would lower the co-occurrence frequency and result in fewer rules found. However, this can be compensated for by lowering the minimum support threshold of the algorithm.

8.2 Building Concept Hierarchies

A natural next step of the work done is to further process the associations found. In the context of ontology learning, the association rules can be used to build concept hierarchies. Sanderson and Croft [1999] defined a method for deciding if a term subsumes (is parent of) another. For two terms, x and y, x subsumes y if the following two conditions hold:

$\mathrm{P}(x|y) = 1$ and $\mathrm{P}(y|x) < 1$, where P is the probability.

This could be applied to the concepts found. In terms of association rules these conditions can be written as:

$confidence(y \Rightarrow x) = 1$ and $confidence(x \Rightarrow y) < 1$.

This in turn, can be written as:

$$\frac{support(y \cup x)}{support(y)} = 1 \text{ and } \frac{support(x \cup y)}{support(x)} < 1.$$

However, this would only include the rules with a confidence of 1. It would therefore be beneficial to reduce the original conditions to $\mathrm{P}(x|y) > \mathrm{P}(y|x)$. This leads to:

$$\frac{support(y \cup x)}{support(y)} > \frac{support(x \cup y)}{support(x)} .$$

By performing this condition on every association rule found, it should be possible to extract the parent concept of a concept. Since a concept most likely is involved in several rules, this method would probably lead to a set of candidate parents for each concept. However, this would probably in most cases be correct.

The implementation of the concept hierarchy has been started on. This is described further in Appendix B.

Appendix A

Acronyms and Abbreviations

API	Application Programming Interface
GATE	Genreal Architecture for Text Engineering
IDI	Department of Computer and Information Science
NLP	Natural Language Processing
NTNU	Norwegian University of Science and Technology
OWL	Web Ontology Language
RDF	Resource Description Framework
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Appendix B

Building Concept Hierarchies

As described in Section 8.2, a natural next step of the work done is to build a concept hierachy based on the association rules found. This work has been started on. A class ParentFinder is found in the ontology module. It takes as a basis the theory presented in Section 8.2. The pseudo code in Figure B.1 shows how this can be implemented.

NounPhrase pseudo code

```
for each Concept
    search for Rule containing Concept as left hand side
    for each Rule found
        currentRule = Rule
        rightHand = right hand of currentRule
        c1 = confidence of currentRule
        search for Rule with Concept as right hand and rightHand
            as left hand
        reflectionRule = Rule found
        c2 = confidence of reflectionRule

        if ( c1 > c2 )
            add right as candidate parent of Concept
        else
            add Concept as candiate parent of rightHand
        end if
    end for each
end for each
```

Figure B.1: Pseudo code of the algorithm for finding candidate parents of a concept.

Note that this work has just been started on and is partly unfinished.

Appendix C

Digital Appendix

Attached to this report is a zip-file containing the following:

- Source code of the implementation.
- The lemmalist used by the lemmatizer module.
- The stop word list used by the index module.
- The two ontologies used for the evaluation.
- The cosine similarity results.
- The results from the evaluation performed by the 4 master students.

References

Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, 1994. URL `http://citeseer.ist.psu.edu/agrawal94fast.html`.

Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 1993. URL `http://citeseer.ist.psu.edu/agrawal93mining.html`.

Hans Olaf Borch. Automatic Keyphrase Extraction, 2005. Project, TDT4730 IDI, NTNU.

Terje Brasethvik. *Conceptual modelling for domain specifi document description and retrieval.* PhD thesis, IDI/NTNU, 2004.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.

M. Delgado, M.J. Martín-Bautista, D. Sánchez, and M.A. Vila. Association Rule Extraction for Text Mining. In *Flexible Query Answering Systems: 5th International Conference, FQAS 2002. Copenhagen, Denmark*, 2002a. URL `http://www.springerlink.com/content/ltj1fv73x7ad30tr/`.

M. Delgado, M.J. Martín-Bautista, D. Sánchez, and M.A. Vila. Mining Text Data: Special Features and Patterns. In *Pattern Detection and Discovery: ESF Exploratory Workshop, London, UK*, 2002b. URL `http://www.springerlink.com/content/ncdk0yk9qj9r43w7/`.

R. Gaizauskas, P. Rodgers, H. Cunningham, and K. Humphreys. GATE User Guide, 1996. URL `http://gate.ac.uk/sale/tao/index.html#x1-40001.2`.

Lee Gillam and Khurshid Ahmad. Pattern Mining Across Domain-Specific Text Collections. In *Machine Learning and Data Mining in Pattern Recognition, 4th International Conference, MLDM 2005, Leipzig, Germany, July 9-11*, 2005. URL `http://www.springerlink.com/content/1vf25nt3a4q4h060/`.

Geir Øyvin Grimnes. Ontologikonstruksjon for Statoil. Master's thesis, IDI, NTNU, 2006.

Thomas R. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human-Computer Studies*, 43(4), 1995.

Jon Atle Gulla. Semantic Technologies at NTNU: Research Challenges and Product Roadmaps. In *Semantiske dager*, 2006.

H. Haddad, J. Chevallet, and M. Bruandet. Relations between Terms Discovered by Association Rules. In *Proc. of PKDD'2000, Workshop on Machine Learning and Textual Information Access, Lyon France*, 2000. URL `http:\www-clips.imag.fr/mrim/User/jean-pierre.chevallet/PUBLICATIONS/HADDAD00a.ps`.

A. Maedche and S. Staab. Semi-automatic Engineering of Ontologies from Text, 2000. URL `http://citeseer.ist.psu.edu/maedche00semiautomatic.html`.

Christopher D. Manning and Hinrich Shutze. *Foundations of Statistical Natural Language Processing.* Massachusetts Institute of Technology, 2002.

M. H Margahny and A. A. Mitwaly. Fast Algorithm for Mining Association Rules. In *AIML 05 Conference, CICC, Cairo, Egypt*, 2005. URL `http://www.icgst.com/AIML05/papers/P1120535119.pdf`.

Kjetil Nørvåg, Trond Øivind Eriksen, and Kjell-Inge Skogstad. Mining Association Rules in Temporal Document Collections. In *Proceedings of the 16th International Symposium on Methodologies for Intelligent Systems (ISMIS'2006), Bari, Italy*, 2006. URL `http://www.idi.ntnu.no/~noervaag/papers/ISMIS2006.pdf`.

Project Management Institute PMI. *A Guide To The Project Management Body Of Knowledge.* Project Management Institute, 2004.

Project Management Institute PMI. *Organizational Project Management Maturity Model.* Project Management Institute, 2003.

Mark Sanderson and Bruce Croft. Deriving concept hierarchies from text. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 1999. ACM Press. URL `http://portal.acm.org/citation.cfm?id=312679&dl=#`.

Geir Solskinnsbakk. Extending Ontologies with Search-Relevant Weights, 2006. Project, TDT4730 IDI, NTNU.

Yasumasa Someya. English lemma list, 1998. URL `http://www.lexically.net/downloads/version4/downloadingBNC.htm`.

Steffen Staab and Alexander Maedche. Ontology Engineering Beyond the Modeling of Concepts and Relations, 2000. URL `http://citeseer.ist.psu.edu/staab00ontology.html`.

World Wide Web Consortium W3C. OWL Web Ontology Language Overview, 2000. URL `http://www.w3.org/TR/2004/REC-owl-features-20040210`.