University of Alberta

Associative Classification, Linguistic Entity Relationship Extraction, and Description-Logic Representation of Biomedical Knowledge Applied to MEDLINE

by

Rafal Rak

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering

Fall 2009
Edmonton, Alberta

Examining Committee

Michel Dumontier, Department of Biology, Carleton University, Ottawa, ON

Lukasz Kurgan, Electrical and Computer Engineering

Marek Reformat, Electrical and Computer Engineering

Osmar Zaïane, Computing Science

Petr Musilek, Electrical and Computer Engineering

Scott Dick, Electrical and Computer Engineering

James Miller, Electrical and Computer Engineering, Committee Chair

To my parents, who have always put the needs of their children before their own, and who have made me the person I am today

Abstract

MEDLINE, a large and constantly increasing collection of biomedical article references, has been the source of numerous investigations related to textual information retrieval and knowledge capture, including article categorization, bibliometric analysis, semantic query answering, and biological concept recognition and relationship extraction. This dissertation discusses the design and development of novel methods that contribute to the tasks of document categorization and relationship extraction. The two investigations result in a fast tool for building descriptive models capable of categorizing documents to multiple labels and a highly effective method able to extract broad range of relationships between entities embedded in text. Additionally, an application that aims at representing the extracted knowledge in a strictly defined but highly expressive structure of ontology is presented. The classification of documents is based on an idea of building association rules that consist of frequent patterns of words appearing in documents and classes these patterns are likely to be assigned to. The process of building the models is based on a tree enumeration technique and dataset projection. The resulting algorithm offers two different tree traversing strategies, breadth-first and depth-first. The classification scenario involves the use of two alternative thresholding strategies based on either the document-independent confidence of the rules or a similarity measure between a rule and a document. The presented classification tool is shown to perform faster than other methods and is the first associative-classification solution to incorporate multiple classes and the information about recurrence of words in documents. The extraction of relations between entities embedded in text involves the utilization of the output of a constituent parser and a set of manually developed tree-like patterns. Both serve as the input of a novel algorithm that solves the newly formulated problem of constrained constituent tree inclusion with regular expression matching. The proposed relation extraction method is demonstrated to be parser-independent and outperforms in terms of effectiveness dependency-parser-based and machine-learning-based solutions. The extracted knowledge is further embedded in an existing ontology, which together with the structure-driven modification of the ontology results in a comprehensible, inference-consistent knowledge base constituting a tangible representation of knowledge and a potential component of applications such as semantically enhanced query answering systems.

Acknowledgments

First and foremost, I would like to express my gratitude to my supervisors Dr. Lukasz Kurgan and Dr. Marek Reformat, for laying a foundation for my research, offering invaluable discussions, and for their guidance during my entire graduate program.

I would like to thank the members of my examination committee, Dr. Michel Dumontier, Dr. Osmar Zaïane, Dr. Scott Dick, and Dr. Petr Musilek, for their involvement and insightful feedback that helped me improve the quality of this dissertation.

I would like to express my appreciation to my lab mates for collaboration, valuable discussions, and for being great everyday companions. Special thanks to Wojciech Stach, who prompted me to commence my PhD program at the University of Alberta in the first place.

I am greatly indebted to Inge Christiaens for biomedical consultation, as well as Justine Gill and Isabelle Sutton for proofreading this dissertation and my other publications.

Last but not least, I would like to thank my family and friends for their constant encouragement and understanding.

Contents

List of Figures

List of Tables

List of publications

The following list enumerates the author's publications pertinent to this dissertation. The numbers correspond to the citations in Bibliography.

[113] Rak, R., Kurgan, L., and Reformat, M. Multi-label associative classification of medical documents from MEDLINE. In *Proceedings of the 4th International Conference on Machine Learning and Applications*, pages 177–184, Los Angeles, CA, USA, December 2005.

[114] Rak, R., Kurgan, L., and Reformat, M. Multilabel associative classification categorization of MEDLINE articles into MeSH keywords. *IEEE Engineering in Medicine and Biology Magazine*, 26(2):47–55, 2007.

[115] Rak, R., Kurgan, L., and Reformat, M. xGENIA: A comprehensive OWL ontology based on the GENIA corpus. *Bioinformation*, 1(9):360–362, 2007.

[116] Rak, R., Kurgan, L., and Reformat, M. A tree-projection-based algorithm for multi-label recurrent-item associative-classification rule generation. *Data and Knowledge Engineering*, 64(1):171–197, January 2008.

[117] Rak, R., Reformat, M., and Kurgan, L. Use of OWL 2 to facilitate a biomedical knowledge base extracted from the GENIA corpus. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (co-located with the 7th International Semantic Web Conference)*, Karlsruhe, Germany, October 2008.

[118] Rak, R., Reformat, M., and Kurgan, L. Extracting functional binary relations between annotated entities in text corpora: a constituent-parser-based approach. *Submitted for publication in Data and Knowledge Engineering*, 2009.

[119] Rak, R., Stach, W., Zaïane, O. R., and Antonie, M.-L. Considering re-occurring features in associative classifiers. In *Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'05)*, pages 240–248, Hanoi, Vietnam, May 2005.

Chapter 1

Introduction

1.1 Motivation

The ever increasing number of scientific articles, which appears in various collections, calls for automated methods that are capable of categorizing, searching, summarizing, and presenting the knowledge embedded therein in a timely manner. For instance, MEDLINE [3], a National Library of Medicine's (NLM) database, which consists of approximately 16 million article references to biomedical journal articles dated back to 1949, is a rapidly expanding collection with currently around 700,000 new article references added every year, which translates to 2,000-4,000 references added each day. Each article reference is semi-automatically assigned terms [19] from Medical Subject Headings (MeSH) [2], a controlled vocabulary that comprises medical terms at various levels of specificity.

Huge datasets pose a challenge to the task of finding relevant information. For instance, PubMed, a publicly available search interface to MEDLINE and other resources, receives over 700 million queries a year[1] from researchers and medical practitioners alike [148]. Search engines performing pure boolean retrieval (returning all documents that match the logical conditions specified in a query) are no longer satisfactory. There is a need for more sophisticated tools capable of determining the relevance of the returned documents with respect to the query, and able to perform semantic inference in order to translate a what-is-stated-like query to a what-is-meant-like query. To some extent this can be accomplished by the use of lexicons, which may extend the query by the synonyms of the words appearing in the query. However, more elaborate solutions involve well-defined concept structures with a network of labeled interconnecting links between the concepts, which could enrich the query by adding some semantic meaning to it in the form of inferred relationships between the concepts specified in the query.

[1] More statistics available on http://www.nlm.nih.gov/bsd/bsd_key.html

These well-defined and well-structured knowledge bases, *ontologies* in particular, not only exhibit powerful reasoning capabilities, but also serve as the visual representation of concisely summarized knowledge that can be tangibly explored by users. Although creating one all-knowing ontology, even limited to a particular field such as biomedicine, is a rather futile task, there are ongoing efforts to build interchangeable links between smaller, specific, and well-established ontologies. For instance, the Open Biomedical Ontologies Foundry (OBO Foundry) [134] is a collaborative project that aims at establishing the framework of interoperable yet orthogonal ontologies in the biomedical domain. The framework consists of ontologies of different levels of specificity forming a pyramid-like structure. Top-level ontologies are well established and subsequently do not change considerably over time; however, specific ontologies constantly evolve with new knowledge and new discoveries regularly appearing in scientific sources. Therefore, not only do the new incoming scientific publications need to be properly categorized, but also the new knowledge contained therein needs to be extracted and embedded into ontologies. This constitutes another task in the field of information retrieval, especially challenging in the biomedicine domain where sentences can be very complex and the language filled with expressions that may be incomprehensible to a layperson.

1.2 Existing solutions

MEDLINE has been attracting researchers attention mainly due to its public availability, large and constantly increasing size, the complexity of specialized, biomedical language, and well indexed and cross-referenced contents. The numerous investigations with the use of MEDLINE include document categorization/classification/indexing to various hierarchical structures or ontologies such as MeSH or Gene Ontology (OBO Foundry candidate) [127, 19, 126, 135, 144, 78], bibliometric analysis [149, 34], semantic (usually ontology-driven) query answering [106, 102, 147], biomedical concept (or named entity) recognition [145, 80, 103], biomedical entity (usually binary) relation extraction [37, 14, 122] as well as more specific protein-protein interaction extraction [69, 45, 152].

Most of the attempts to categorize documents are based on document feature extraction and the use of machine learning methods to learn the features and apply the learnt model to new, unseen documents [87, 127, 135]. However, although the machine learning tools used in this process possess the predictive power, they lack the descriptive value, i.e., the models are not “transparent” to a user. Additionally, very effective state-of-the-art binary classifiers do not scale well when applied to the classification problem, which requires assignment of multiple labels to a single document [144].

Similar techniques, i.e., the use of machine learning, is used in relation extraction [22]. The

features used in the learning process are mainly based on the output of *shallow* or *deep linguistic parsers*. However, the effectiveness of this approach is inferior to methods that rely on manually tailored sets of patterns or rules that are matched against the output of *typed dependency* parsers, which produce *grammatical dependencies* between tokens in a sentence, to extract relations without the learning process [36, 122]. These methods, on the other hand, lack a standardized set of grammatical dependencies, which makes them sensitive to the selection of the parser.

1.3 Thesis statement and contributions

In this work several novel methods that contribute to the tasks of document categorization and relation extraction are proposed.

In the case of categorizing documents, special emphasis is put on techniques capable of building *descriptive* models, which can be easily interpreted by humans, an important characteristic that allows diagnosing the means that led to the prediction in a comprehensible manner. Additionally, as opposed to binary classifiers, a single model is expected to incorporate all the classes to overcome the problem of scalability when applied to classification with a large number of classes. In the case of relation extraction, the problem of parser-dependence is overcome by using a standardized set of tags used in *constituent trees*, which represent the syntactic decomposition of a sentence. Due to the fact that the proposed method uses similar principles to other pattern/rule-based methods, it is expected to maintain superior effectiveness when compared to the machine learning techniques.

As a result, this work proposes (1) a fast algorithm for generation of multi-label document classification models based on the descriptive *associative classification* approach, and (2) an algorithm for extracting relations between biological terms in documents based purely on the syntactic decomposition of constituents in sentences. Additionally, the structural modification and enrichment of an existing ontology is shown as an application that embeds the knowledge discovered in the process of extracting relations from documents. Such an ontology concisely summarizes the discovered knowledge and may be further used in, e.g., semantic query answering systems.

To summarize, this dissertation lays forward the following thesis statements:

1. Associative classification with recurrent items is an efficient and human-interpretable method to categorize large text data with multiple labels.

2. Constituent trees are sufficient representations of sentences to effectively extract a variety of relationships that appear in linguistically complex sentences.

3. Web Ontology Language (OWL) is a highly expressive, description-logic-based language capable of incorporating automatically extracted textual knowledge in a reasoner-oriented manner.

1.4 Overview of the proposed approach

The objectives of the various investigations presented in this dissertation include:

Classification A fast tool for building descriptive models capable of classifying text documents to multiple categories.

Relation extraction An effective tool for extracting user-defined relations between annotated concepts in text.

Ontology enrichment Structural, conceptual, and quantitative modification of an existing ontology based in part on knowledge discovered in the process of relation extraction.

An application scenario that incorporates the three objectives is shown in Figure 1.1.

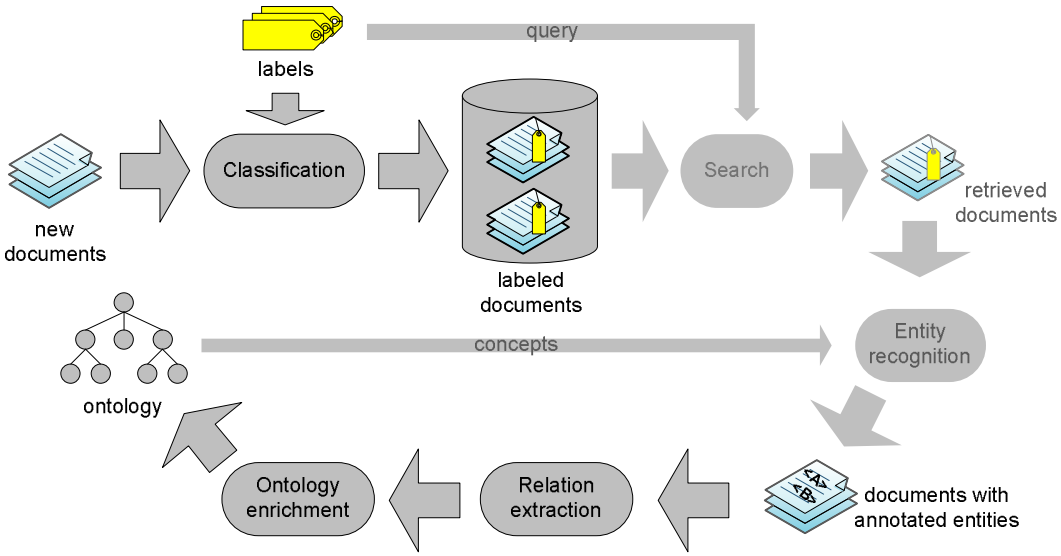


Figure 1.1: Application scenario of classification, relation extraction, and ontology enrichment components.

The scenario involves classification of new, uncategorized documents to a set of predefined labels to form a well-indexed repository of documents. A portion of the labeled documents is retrieved for further investigation, this time narrowed to a specific field of study or a specific ontology. This portion of documents is processed to annotate entities in text based

on concepts from the ontology. The relation extraction process captures relations between these entities and populates the ontology with the newly discovered knowledge.

Relating the diagram to the data sources and knowledge bases used in this work the scenario involves the use of (1) MEDLINE as the source of new documents, (2) MeSH as the set of labels, (3) the GENIA corpus as an example of a collection of documents retrieved from MEDLINE using a query composed of several MeSH terms, and (4) the GENIA ontology that serves as both the source of concepts to annotate entities in text and the destination of the extracted relations (see Section 2.2.1 for the description of the aforementioned data sources). It is important to notice that the GENIA corpus already includes manually identified entities annotated according to the GENIA ontology, thus this information is used directly in the process of relation extraction.

The following sections briefly summarize the work presented in Chapters 3, 4, and 5.

1.4.1 Multi-label associative classification

The classification part is based on associative classification, which enables building descriptive models that, apart from their predictive capabilities, provide easily interpretable rules. The rules are accompanied by the level of confidence which they hold in the given dataset. The proposed solution is an *Apriori*-like approach with novel techniques of enumerating itemsets and projecting transactions (see Section 2.1.1 for an introduction to text classification and associative classification in particular).

As opposed to binary classifiers that require independent training for each class when applied to multi-label classification, the proposed algorithm produces a single model that incorporates all the classes. This significantly reduces the time needed to build the model, especially important in classification with large number of classes. The algorithm is also capable of accounting for the recurrence of features (words) in documents. Evaluation comprises a scalability study as well as the effectiveness of classification with different thresholding strategies. Although the comparative evaluation shows that state-of-the-art Support Vector Machines (SVM) remains superior in terms of classification effectiveness, it is demonstrated that this method is not efficient in the multi-label classification environment, unlike the proposed associative classifier.

The findings on multi-label, recurrent-item, associative classification, presented in Chapter 3, have been published in parts in [119, 113, 114, 116] with Lukasz Kurgan and Marek Reformat as major collaborators, as well as Wojciech Stach, Osmar Zaïane, and Luiza Antonie, who were involved in the investigation of the recurrence of items in transactions. RR conceptualized, designed, and implemented the algorithm for multi-class, recurrent-item rule generation, prepared the datasets, performed the evaluation of efficiency and effectiveness

of multi-class classification, analyzed the results, and wrote the manuscripts [113, 114, 116] and helped with writing the manuscript [119]. LK and MR helped with conceptualization of [113, 114, 116], coordinated the projects, and helped with writing the manuscripts [113, 114, 116]. OZ and LA conceptualized and coordinated the project investigating the recurrence of items, performed experimental evaluation, analyzed the results, and wrote the manuscript [119]. WS helped with implementation of the algorithm for recurrent-item rule generation and with writing the manuscript [119].

1.4.2 Entity relation extraction

The proposed method for extracting relations between annotated entities in text is based on constituent trees and involves a set of manually tailored patterns representing different types of relations (see Section 2.1.2 for an introduction to text mining and natural language processing). Experiments performed on a set of sentences from the GENIA corpus processed by two different constituent parsers, the Stanford parser and the Charniak-Lease parser, show that the method outperforms other approaches based on shallow parsers and dependency parsers. The constituent pattern trees are shown to be parser-independent and flexible enough to cover a vast majority of the different types of relations while maintaining high levels of both precision and recall while yielding almost perfect scores on a subset of error-free constituent trees. Similar results with the same set of patterns were obtained from a general English corpus showing that the developed patterns can be used across different application areas.

The investigation of constituent-tree-based relation extraction presented in Chapter 4 was undertaken in collaboration with Marek Reformat and Lukasz Kurgan and has been submitted for publication in an international journal [118]. RR defined the problem of constituent tree inclusion, conceptualized, designed, and implemented the algorithm for solving the problem, prepared the datasets, performed the evaluation of efficiency and effectiveness of the method, analyzed the results, and wrote the manuscript. MR and LK helped with conceptualization of the solution, coordinated the project, and helped with writing the manuscript.

1.4.3 Ontology enrichment

The GENIA ontology enrichment (or population) is demonstrated as an application of utilizing knowledge discovered in the process of relation extraction. Moreover, the transformation of the ontology to conform with OWL and its expressive description logic is investigated (see Section 2.1.3 for an introduction to knowledge representation, ontologies, and OWL). As a result, a new, structurally changed ontology is presented. The ontology

encompasses a more comprehensive (reasoner-oriented) taxonomy of categories, relationships between biological entities, and a hierarchy of relationships. OWL proved to be a well-suited language to accommodate the knowledge base.

The experiences of using OWL to encode the various new features of the enriched GENIA ontology, presented in Chapter 5, have been published in parts in [115, 117] in collaboration with Lukasz Kurgan and Marek Reformat. RR conceptualized the problem, prepared tools for creating the ontologies, created the ontologies, and wrote the manuscript. LK and MR helped with conceptualization of the problem, coordinated the project, and helped with writing the manuscript.

1.5 Outline

The remainder of this dissertation is presented as follows. Chapter 2 includes background information necessary to understand the issues discussed in the remaining chapters. A special emphasis is put on the role of associative classification in text categorization, deep linguistic parsing of sentences in relation extraction, and OWL as an ontology language in knowledge representation. Chapters 3, 4, and 5, present the main contributions and correspond to the multi-label associative classification, biological entity relation extraction, and ontology enrichment projects, respectively. The dissertation is summarized by enumerating contributions and findings as well as limitations and future directions in Chapter 6.

Chapter 2

Background and Related Work

2.1 Background

The following sections include background information necessary to understand the issues discussed in the coming chapters. Section 2.1.1 describes text categorization (classification) focused especially on associative classification and thresholding strategies used in multi-label classification problems. The methods of extracting knowledge from textual sources are discussed in Section 2.1.2, whereas Section 2.1.3 provides a concise introduction to formal and practical notations used in knowledge representations, ontologies in particular. The three sections provide background for Chapters 3, 4, and 5, respectively.

2.1.1 Text classification

Text classification or *text categorization*[1] is one of the task of *Information Retrieval*, i.e., searching for information contained in or based on documents. The goal of text classification is to label text with thematic categories from a predefined set [131].

More formally, given a set of documents $\mathcal{D} = \{d_1, \ldots, d_{|\mathcal{D}|}\}$ and a set of categories $C = \{c_1, \ldots, c_{|C|}\}$, categorization aims at assigning a Boolean value (*true* or *false*) to each pair $\{d_j, c_i\} \in \mathcal{D} \times C$. The task is to find a hypothesis (model), $\mathcal{H} : \mathcal{D} \times C \rightarrow \{true, false\}$, that approximates the unknown *target function*, i.e., a function that describes how the documents should be classified.

Text categorization appears in many varieties by imposing constraints on the definition

[1] The terms *classification* and *categorization* are used interchangeably. Although the term *annotation* is also used in literature to describe classification/categorization, here *annotation* is used exclusively to denote the marking of one or more words in text and should not be mistaken with *classification/categorization* meant as assigning labels to a document.

provided above. *Binary categorization* limits the number of predefined categories to two, c and its complement $\overline{c}$, and therefore can be simplified to hypothesis $\mathcal{H} : \mathcal{D} \rightarrow \{true, false\}$. *Single label categorization* is when exactly one category from a set of $|C|$ categories can be assigned to each document $d_j \in \mathcal{D}$. Lastly, *multilabel categorization* is when any number of $|C|$ categories can be assigned to each document.

Research on building hypotheses that would best approximate target functions has been carried out for several decades. The most popular and accurate methods come from *machine learning* [99], where *inductive learning* is utilized in the process of building a hypothesis. Inductive learning in text categorization is a process that "observes" certain characteristics of a set of pre-categorized documents, e.g., frequently co-occurring words, based on which a hypothesis is built. It is expected that the discovered characteristics be present in unseen (yet to be categorized) documents.

The machine learning and data mining methods of text categorization include instance-based classifiers [87, 94], probabilistic (based on Bayes' theorem) classifiers [85, 88, 91, 125, 94], neural networks [47, 127, 61], support vector machines [70, 71, 96, 57, 157], decision trees [108, 94, 25], decision rules [42, 154, 119, 114, 111], associative classifiers [95, 155, 154], and other.

Associative classification, a promising *descriptive* method of classification, is an approach that is investigated in this work and presented in Chapter 3. The next section provides an introduction to the problem of associative classification.

Associative classification

Associative classification is one of the classification methods that produces a *descriptive*, human-interpretable model. Associative classification is a simple reformulation of *association rule mining*, a process of discovering *frequent* patterns in data and forming a set of rules based on those patterns. The problem of association rule mining was first defined by Agrawal and Srikant [16] who used an analogy to *shopping basket analysis*. Therefore, the original terminology, such as *transactions* and *items*, has been used ever since and is kept throughout this paper. In the context of text classification, transactions refer to documents, whereas items refer to words in these documents.

Formally, the problem is defined as follows. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of items (an alphabet). Let D be a set of n transactions T_j, $\forall j \in [1, n] : T_j \subseteq \mathcal{I}$, such that $\bigcup_{j=1}^{n} T_j = D$. The goal of association rule mining is to find a *frequent pattern* or *frequent itemset* $X = \{i_{x1}, i_{x2}, \ldots, i_{xp}\}$ in transactions T_j such that X is contained in at least $\hat{\sigma}$ transactions T_j. $\hat{\sigma}$ is called a *support count* and is one of the parameters used in the association rule mining. More often it is expressed as the fraction of transactions in D that contain pattern X, or

formally:

$$\sigma_D(X) = P(X) = \frac{\hat{\sigma}_D(X)}{|D|} \tag{2.1}$$

where $\hat{\sigma}_D(x)$ denotes the support count of x in D.

The rules are obtained from frequent patterns by splitting them into two parts, antecedent X_A and consequent X_C, where $X_A \cap X_C = \emptyset$. The rule of the form $X_A \rightarrow X_C$ indicates that the transactions containing X_A also contain X_C with probability φ in D. φ is called *confidence* and is formally defined as follows:

$$\varphi_D(\langle X_A, X_C \rangle) = P(X_C|X_A) = \frac{P(X_A, X_C)}{P(X_A)} = \frac{\hat{\sigma}_D(\langle X_A, X_C \rangle)}{\hat{\sigma}_D(X_A)} \tag{2.2}$$

The association rule mining process, in its basic form, is subjected to two thresholds, minimum support and minimum confidence. Both these values are treated as parameters in the process of association rule mining.

In associative classification the above problem is reformulated such that the goal is to find association rules in the form of $X \rightarrow C$, where X is a set of classification features and C is a set of class labels. The intuition behind associative classification is that the rules $X \rightarrow C$, which constitute the classification model, indicate strong relationship between items in X and the set of classes C. Therefore, new and unlabeled documents that consists of the set of items X are likely to be labeled with the set of classes C.

An associative classifier is presented in Chapter 3 and applied to categorization of MEDLINE documents to multiple MeSH terms (see the description of MEDLINE and MeSH in Section 2.2.1). The next section discusses strategies commonly used in deciding on class assignment in the multi-label environment.

Thresholding strategies in multi-label classification

Apart from a decision whether a particular class should be assigned to a document, classifiers also produce a score (confidence or probability), which is a numerical value that indicates how well a class fits the document being classified. In multi-label classification, scores for each class–document pair are the basic measures used in *thresholding strategies* while deciding on class assignment.

There is a variety of thresholding strategies, which can be grouped in the following categories [151]:

RCut For each *document*, the *ranked-based* strategy involves sorting classes according to

their scores and choosing the top t classes, which will constitute the classification decision for that particular document.

PCut For each *class* $c \in C$, the *proportional-based* strategy involves sorting the documents according to their scores and assigning $k = P(c) \cdot x \cdot |C|$ top documents to c, where $P(c)$ is the fraction of documents assigned to class c in the training set, $|C|$ is the number of classes, and x is a real-valued parameter that may range from 0 (in which case no documents will be assigned to c) to the total number of documents $|\mathcal{D}|$ (in which case all documents will be assigned to c).

SCut The *score-based* strategy assigns a class to a document based purely on the score between the two and depends on a score threshold s_i, which is parametrized separately for each class.

The RCut parameter t and the set of SCut parameters $\{s_1, \ldots, s_{|C|}\}$, are tuned on a *validation* set (see Section 2.2.2). RCut t as well as PCut x may also be specified by a user, in which case x should be set close to the average number of documents a classifier assigns to a class.

The three main strategies have their pros and cons. Yang [151] points out, for instance, that although PCut uses the most information (class distribution) to make classification decisions, it is not suitable for online responses, i.e., it is not able to make the decisions for each document separately, which is a major drawback in real-life scenarios, such as categorization of MEDLINE to MeSH (which is one of the main foci of this dissertation), that perform classification instantly every time a new document arrives. SCut per-class tuning may overfit the validation set, which is less likely in case of RCut as it uses only a single parameter. On the other hand, properly cross-validated SCut appears to be the best choice for online classification as each class is parametrized separately. Such individual class-oriented parametrization, however, may become an inconvenient solution when a large number of classes is considered (as is the case with MeSH).

2.1.2 Biomedical text mining

Biomedical text mining is a process of deriving information from biomedical text and constitutes an important part of a broader discipline, text mining. Biomedical text mining mainly includes biomedical *named entity recognition* and *relation extraction*.

Named entity recognition is a process of identifying biological terms (entities) in text, whereas relation extraction aims at capturing some kind of activity between (usually) two entities. For example, the sentence *"Spi-B binds DNA sequences containing a core 5'-GGAA-3' and activates transcription"* contains four entities, *Spi-B*, *DNA sequences*, *5'-*

GGAA-3', and *transcription*, and three relations, *Spi-B binds DNA sequences*, *DNA sequences contain 5'-GGAA-3'*, and *Spi-B activates transcription*.

Various approaches have been introduced for both biological entity recognition and relation extraction between those entities. These attempts can be classified into the following groups:

Linguistic-based approach. In the case of entity recognition, this approach often involves natural language processing (NLP) and is often seen as a preliminary step used in other methods [18, 103, 38, 37, 14]. In the case of relation extraction, it is reduced to employing shallow parser techniques that, based on some pre-developed patterns, search for a certain sequence in text.

Dictionary-based approach. Biological terms are identified by scanning text and matching expressions with dictionary entries [74, 141, 103, 145, 38]. The drawback of this solution is that a dictionary may not contain all variations of spelling and thus, additional linguistic-based preprocessing is usually required.

Machine-learning-based approach. This approach is mainly used in biological term recognition and includes machine-learning methods [80, 21, 22]. Learning process is usually based on training sets consisting of pre-generated *n-grammes* from a corpus and mostly character-based and word-based features are taken into consideration.

Statistical approach. This approach, found in biological term recognition as well as term relations extraction, involves the use of statistical tools and is mostly based on counting co-occurrence of sequences of words in text [38, 37, 14]. This information may be further used to identify terms and relations as well as to calculate their strength.

Due to their complementary properties, most of the recent systems use a combination of the approaches. Selected achievements in biomedical text mining are presented in Section 2.3.

The proposed relation extraction method presented in Chapter 4 is a linguistic-based approach, which heavily relies on natural language processing described in the following section.

Natural language processing

Natural language processing (NLP) is a broad concept that, in information retrieval, can be roughly summarized as converting text into a formal, computer-understandable representation. In the context of text mining (as described in this section), NLP usually involves part-of-speech (POS) recognition and shallow or deep linguistic parsing including *constituent parsing* and *dependency parsing*.

Shallow parsers, or chunkers, aim at finding contiguous, non-overlapping spans of words by grouping them into *chunks*, presumably atomic grammar structures. They work on POS-tagged sentences and are not guaranteed to group all of the words in a sentence, i.e., some words may not belong to any chunk. An example of a popular chunker is CASS [11].

Constituent parsers (or phrase structure parsers) process sentences using the productions of a grammar and produce one or more multi-level tree structures per sentence. An example of a constituent tree is given in Figure 2.1. Due to the many ambiguities in languages, the parsers produce several alternative trees accompanied by probabilities with which the trees are likely to be correct decompositions of the sentences they represent. Usually only the tree with the highest probability is taken into further consideration. Some notable examples of constituent parsers include the Charniak parser [32], the Stanford parser [82], the Bikel parser [27], and the Collins parser [43].

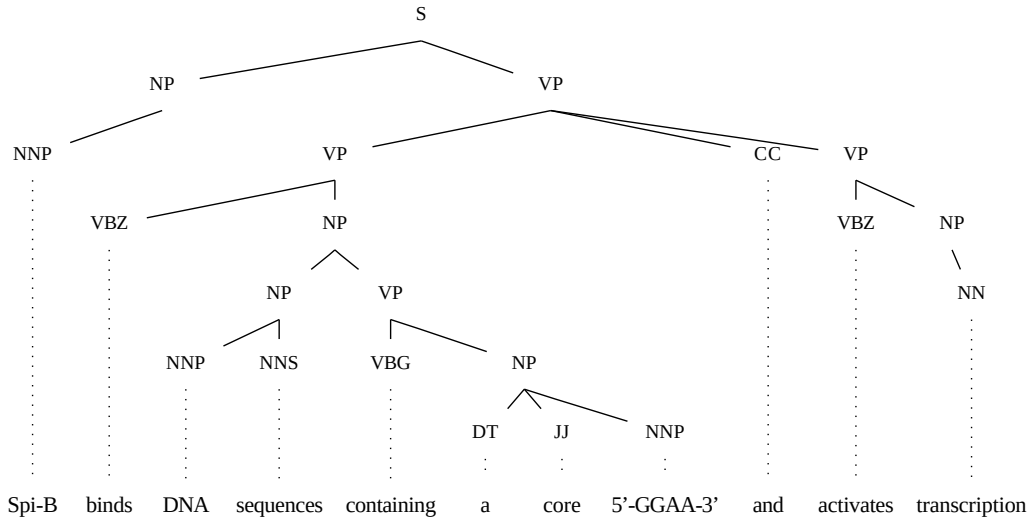


Figure 2.1: Example of a constituent tree. The POS and constituent labels correspond to the Penn Treebank notation [26] (see Appendix A for the explanation of the tags).

The typed dependency parsers aim at assigning a grammatical dependency/role (such as *subject*, *object*, *modifier*, etc.) between a pair of tokens. This is accomplished by (1) identifying *heads* and their *dependents* and (2) applying a pattern search algorithm that matches a grammar relation to each head-dependent pair identified in the previous step. The first step is usually based on a constituent tree, whereas the second step involves the usage of manually prepared grammar relation patterns that match phrase structures with the grammar relations. An example of a dependency graph is given in Figure 2.2. Some examples of typed dependency parsers include the Link parser [133], Minipar [51], and the Stanford dependency parser [49] (an extension of the Stanford constituent parser [82]). The main differences between these parsers lie in the number and the structure of the considered

grammatical relations.

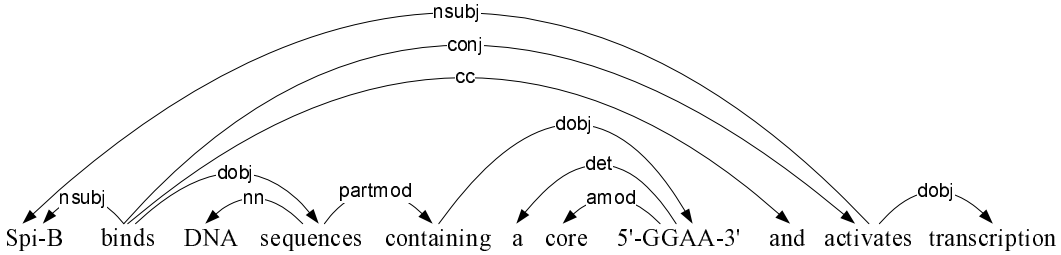


Figure 2.2: Example of a dependency graph. The dependency role labels correspond to those produced by the Stanford dependency parser [49] (see Appendix B.2 for the explanation of the role labels).

Chapter 4 discusses the use of the three parsers in the process of relation extraction.

Regular expressions

Extracting or transforming textual information often involves the use of *regular expressions*, concise sequences of characters that form flexible patterns capable of capturing a broad range of characters, words, or larger constituents in text.

Regular expressions were introduced by Kleene [81] and were originally applied to automata theory. Kleene described models of automata using his own mathematical notation originally called *regular sets*. Later on these sets were implemented in text editors and used to match patterns in text. Nowadays, the regular expressions appear as a built-in functionality of many programming languages such as Perl, Ruby, Python, Tcl, Java, JavaScript, and PHP.

Regular expressions define sets of strings (constants) and operations over these sets. Formally, given two sets of strings P and Q over some finite alphabet the following three operations are defined:

- *Product* or *Concatenation*. $PQ = \{pq | p \in P \wedge q \in Q\}$.
- *Union* or *Alternation*. $P|Q = P \cup Q$.
- *Kleene star* or *Iteration*. $P^* = \bigcup_{n=0}^{\infty} P^n$, where $P^2 = PP$, $P^3 = PPP$, etc., and $P^0 = \varepsilon$, where ε denotes an empty string.

If there is ambiguity between operations in a regular expression, i.e., parentheses grouping operations are omitted, iteration has the highest priority followed by concatenation and alternation. Several examples or regular expressions are shown in Table 2.1.

Table 2.1: Examples of regular expressions

Regular expression	Matching sets of strings
`ab`	$\{ab\}$
`a|b`	$\{a, b\}$
`a*`	$\{\varepsilon, a, aa, aaa, \ldots\}$
`a*|b`	$\{\varepsilon, a, b, aa, aaa, \ldots\}$
`(a|b)*`	$\{\varepsilon, a, b, aa, aa, aaa, \ldots, b, bb, bbb, \ldots\}$
`ab|c(de)*`	$\{ab, abde, abdede, abdededede, \ldots, ac, acde, acdede, acdededede, \ldots\}$

Regular expressions are used extensively in this work, e.g., in preprocessing of text corpora to desired formats. They are also a means for creating relation extraction patterns as described in Chapter 4.

In this dissertation, regular expressions are presented in a commonly used (especially in the software developers community) notation (supported by languages such as Perl, Java, Tcl, etc.). Some frequently used examples of regular expressions are given in Table 2.2.

Table 2.2: Examples of Perl-like regular expressions

Regular expression	Matching strings
`/.*/`	any (possibly empty) string
`/abc/`	any string containing *abc*
`/^abc/`	strings that begin with *abc*
`/xyz$/`	strings that end with *xyz*
`/^abc|xyz$/`	strings that begin with *abc* or end with *xyz*
`/^(abc)?xyz$/`	*abcxyz* and *xyz* only

2.1.3 Knowledge representation

Ontology and the Semantic Web

Ontologies, a key enabling technology for the Semantic Web [24], were developed in artificial intelligence to facilitate knowledge sharing. Nowadays they are utilized in fields such as intelligent information integration, information retrieval, electronic commerce, and knowledge management [48]. Biomedicine is one of the areas where large and standardized structured vocabularies, e.g., OBO Foundry [134], are being developed.

Two main ontology layers in the context of the Semantic Web applications include the definition layer and the instance layer. The definition layer represents the structure of the ontology and provides the definition of the ontology concepts described by a set of properties. Once the definition layer is constructed, the ontology is populated with the real data that constitute the instance layer.

One of the most important aspects in the realization of the Semantic Web is the development of languages to encode knowledge in order to make it comprehensible for web agents searching for information. World Wide Web Consortium (W3C)[2] has developed RDF [7], an assertion-based language intended to provide a basic foundation for the more advanced languages of knowledge representation, such as DAML+OIL [1] or OWL [6]. These languages have a well-defined semantics and are capable of manipulating complex taxonomic relations between entities on the Web [98].

Description logic

Description Logic (DL) is a knowledge representation language that facilitates creating, reasoning about, and manipulating knowledge bases [20]. A knowledge base expressed in DL consists of two components:

TBox The TBox contains *terminology*, which is comprised of *concepts* denoting sets of individuals (e.g., `Person`, `Woman`), and *roles* denoting binary relationships between individuals (e.g., `hasChild`), as well as recursive descriptions for defining complex concepts from atomic concepts and roles (see below).

ABox The ABox contains *assertions* about individuals in the scope of the TBox (e.g., "John" and "Jane" may be asserted as individuals of classes `Man` and `Woman`, respectively).

DLs come in many varieties, which differ in their *expressiveness*. Usually the more expressive the DL, the more complex (less tractable) the inference becomes. For instance, the $\mathcal{AL}$ DL, which is considered the basic DL, includes the following (informally stated) definitions:

- atomic concept A,
- top concept $\top$, which covers all possible individuals,
- bottom concept $\bot$, which has no individuals,
- complement of an atomic concept $\neg A$,
- intersection of two concepts C and D, denoted $C \sqcap D$,
- universal quantification (value restriction), denoted $\forall R.C$, where R is a role and C is a concept,

[2] http://www.w3.org

- limited existential quantification, denoted $\exists R.\top$, where R is a role.

For example, the following statements can be expressed in $\mathcal{AL}$: $\texttt{Person} \sqcap \neg \texttt{Woman}$, $\texttt{Woman} \sqcap \exists \texttt{hasChild}.\top$, and $\texttt{Woman} \sqcap \forall \texttt{hasChild}.\texttt{Man}$, which denotes persons who are not women, women who have at least one child, and women all of whose children (if any) are men, respectively. On the other hand, the union $\texttt{Man} \sqcup \texttt{Woman}$ is an example of a statement that cannot be expressed in $\mathcal{AL}$.

Using the equivalence '$\equiv$' and inclusion '$\sqsubseteq$' operators the following complex descriptions can be created: $\texttt{Man} \equiv \texttt{Person} \sqcap \neg \texttt{Woman}$, i.e., men *are equivalent to* persons who are not women; $\texttt{Man} \sqsubseteq \texttt{Person}$, i.e., men *are* persons or, in other words, Person is a more general concept that Man; and $\texttt{Man} \sqcap \texttt{Woman} \sqsubseteq \bot$, i.e., men and women are two distinct (not sharing any individuals) concepts.

Equivalence and inclusion have two distinct interpretations. Equivalence ($A \equiv B$) satisfies the *necessary* and *sufficient* criteria (i.e., if an individual is a member of class A, it is also a member of class B, and vice versa), whereas inclusion ($A \sqsubseteq B$) satisfies only the *necessary* criteria (i.e., if an individual is a member of class A, it is also a member of class B, *but* the reverse does not hold).

Description Logic forms a base for many ontology languages such as OWL (described in the next section).

Web Ontology Language (OWL)

Web Ontology Language (OWL[3]) [6] is a widely used ontology standardized specification popularized in both academic and commercial sectors [132, 54]. The language is heavily based on Description Logics, and thanks to its explicit logical basis successfully superseded less formal specifications such as DAML+OIL. The original version of OWL, or more specifically its sublanguage, OWL-DL, provides the expressiveness of the $\mathcal{SHOIN}(D)$ DL language, whereas its successor OWL 2 [4] corresponds to the $\mathcal{SROIQ}(D)$ logic.

The expressiveness of OWL 2[4], $\mathcal{SROIQ}(D)$, includes:

- the $\mathcal{AL}$ DL (described in the previous section) augmented by qualified existential quantification ($\exists R.C$), concept union ($C \sqcup D$), and *complex* concept negation ($\neg C$),
- complex roles inclusion ($R_1 \circ \cdots \circ R_n \sqsubseteq S$),
- qualified cardinality restrictions ($\geq n\ R.C$, $\leq n\ R.C$),

[3] The natural acronym WOL was intentionally replaced by its creators with easier to pronounce and associate with, OWL.

[4] As of 22 September 2009, the OWL 2 specification is a W3C Proposed Recommendation.

- inverse, transitive, reflexive, irreflexive, and disjoint roles,
- data values, data types, and data roles.

OWL uses its own jargon and thus concepts and roles are substituted with *classes* and *properties*, respectively. It also differentiates between *object properties* (roles between concepts) and *data properties* (roles between concepts and data)[5].

The primary syntax of OWL is RDF/XML [7] and is meant to be used for exchanging ontologies written in OWL 2 among tools and applications. Additionally, OWL 2 provides more human-readable syntaxes, such as the functional-style syntax and the Manchester syntax, all of which can be translated to the primary RDF/XML syntax[6].

For example, the axiom $\texttt{Person} \sqcap \geq 2\ \texttt{hasChild}.\texttt{Man} \sqcap \leq 3\ \texttt{hasChild}.\texttt{Woman}$, which denotes persons who have at least two sons and at most three daughters, is written in the OWL functional-style syntax as:

```
ObjectIntersectionOf(
  Person,
  ObjectMinCardinality(2 hasChild Man),
  ObjectMaxCardinality(3 hasChild Woman))
```

The functional-style syntax is used in Chapter 5 to encode the proposed enrichment of an existing ontology in OWL 2.

Ontology evolution

Ontology evolution is the process of encompassing a set of activities that ensure timely adaptation of an ontology to changes, and propagation of those changes to dependent objects wile preserving the consistency of underlying data [139]. Ontology evolution is realized by the means of *ontology changes* and involves defining a set of possible changes (heavily dependent on the structure of an ontology) as well as decisions on how and when to introduce these changes in order to preserve consistency in the underlying structure of the ontology, its instances, applications, and other knowledge sources dependent on the ontology being modified.

Stojanovic [139] identifies the following types of change discovery:

Structure-driven change discovery involves the decisions to modify the ontology's structure and is handled by ontology engineers.

[5] The use of data values, data types, and data properties is denoted by the '(D)' in $\mathcal{SROIQ}(D)$.

[6] *Protégé* (http://protege.stanford.edu) is an example of a popular tool for ontology management that supports all RDF/XML, funtional, and Manchester syntaxes

Data-driven change discovery involves changes in the underlying data (instances) in order to refine an ontology.

Usage-driven change discovery is the process of adapting the ontology to end-users' needs, e.g., by tracking frequency of accessing certain parts of the ontology, which may become a signal for engineers to rearrange those parts.

This work presents the structure-driven and data-driven change discovery, as described in Chapter 5, to modify the structure of an existing ontology as well as to enrich the ontology with knowledge extracted from text.

2.2 Performance evaluation

This section describes datasets and data structures together with effectiveness measures and evaluation techniques that were used to test the various methods presented in the following chapters.

2.2.1 Datasets

The OHSUMED and GENIA corpora [67, 79] (presented below in detail) are the main biomedical datasets used in experiments. The remaining datasets serve to show effectiveness and efficiency of the presented methods in a broader, non-biomedical context.

Both OHSUMED and GENIA are subsets of the MEDLINE database, where OHSUMED is a complete collection of articles with *MeSH keywords* (a set of well-defined, structured biomedical categories) from a certain period of time, and GENIA is a selected collection of articles matching a small set of *MeSH keywords*. The two collections differ in size and in terms of additional information associated with them, and therefore serve different purposes. The OHSUMED corpus together with the MeSH categories are used in Chapter 3 to verify classification capabilities of the presented methods, whereas the GENIA corpus and the *GENIA ontology* are used in Chapters 4 and 5 to demonstrate biological relation extraction and ontology enrichment process.

Additionally, RCV1 [90], a vast collection of Reuters articles, is used to show the scalability of the proposed algorithm for generating multi-label, associative-classification rules (see Chapter 3), whereas a general English corpus [22] is used to demonstrate the diverse domains the proposed algorithm for relation extraction from text can be applied to (see Chapter 4).

OHSUMED

The OHSUMED collection [67] consists of 348,566 records from MEDLINE, a National Library of Medicine's (NLM) database consisting of approximately 16 million article references to biomedical journal articles. OHSUMED is limited in scope to 5 years, 1987 to 1991. Each article includes title, abstract, 10 to 15 MeSH indexing terms, author, source, and publication type. A modified version of the dataset used in this work is limited to 233,445 documents and comprises the original OHSUMED articles that have both titles and abstracts. This version has become a standard in reported text classification attempts [92, 150, 127].

MeSH is an annually updated controlled vocabulary of medical terms [2]. The thesaurus consists of over 25,000 terms arranged in an eleven-level hierarchical structure[7] corresponding to the various levels of specificity. At the top of the tree structure there are 15 general concepts such as *Anatomy*, *Organisms*, or *Diseases*. Any other lower-level concept can occur more than once in the tree. A fragment of the MeSH hierarchy is shown in Figure 2.3.

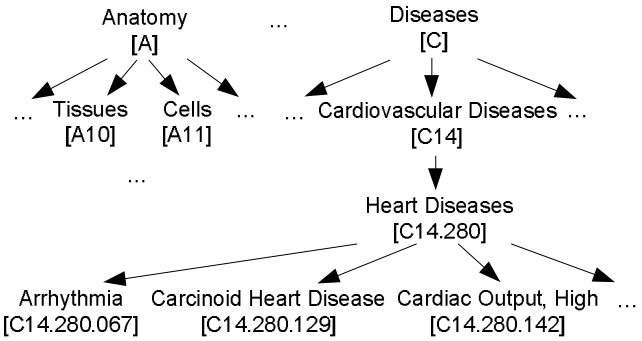


Figure 2.3: Fragment of the MeSH hierarchy

GENIA corpus and ontology

The GENIA corpus [79] consists of a set of 2000 annotated abstracts from the MEDLINE database. This subset was obtained by querying the database with the MeSH terms, *human*, *blood cell*, and *transcription factor*. The annotation includes, but is not limited to, sentence boundaries, term boundaries, and the classification of biological entities. GENIA consists of 18,545 sentences, which contain a total of 96,582 annotated biological entities (including nested entities). The biological entities are categorized according to the GENIA ontology, a taxonomy of 47 categories also created by the authors of the corpus.

Since its development, both the corpus and the ontology have been intensively used by

[7] The numbers provided for both MeSH and MEDLINE correspond to the 2009 release

researchers in biological entity recognition [80, 157], query answering [136, 12, 14], and ontology creation and population [128, 13].

An example of a MEDLINE document and its annotated version in the GENIA corpus are shown in Figure 2.4.

Each document in GENIA has a MEDLINE Unique Identifier[8], title, and abstract. *Biological entities*, i.e., multi-word expressions that carry some biologically significant meaning, such as *IL-2 gene*, *nuclear protein*, *T cell activation*, etc., are enclosed in (possibly nested) `cons` tags that are further enclosed in `sentence`, `abstract` or `title`, and finally, `article`.

Each of the entities is assign one of 36 distinct categories. These categories together with additional 12 concepts that generalize them constitute the GENIA ontology[9]. The ontology is simply the hierarchy of categories, which begins with top three concepts, `Source`, `Substance`, and `Other_name`, and is distributed across six levels (`Other_name` being a sole example of a terminal category on the first level. A fragment of this ontology is shown in Figure 5.2(a) in Chapter 5.

RCV1

RCV1 [90] is a collection of news stories from Reuters, which consists of 804,414 records covering exactly one year of publishing limited to the English version only. The stories are manually assigned labels from three sets of categories: 103 topic labels, 354 industry labels, and 366 region labels. Both the topic and industry labels are organized in hierarchical structures denoting their level of specificity.

The collection has mainly been used in testing machine learning methods applied to classification [90, 72, 60].

General English corpus

The general English corpus, as it is being referred to in Chapter 4, is a collection of 500 sentences used in the open information extraction project [22]. The corpus consists of mostly article headlines and biographical entries, and comes with annotated entities such as the names of companies, people, and places. Each sentence consists of exactly two manually annotated entities.

[8] In 2004 MEDLINE Unique Identifier (UI) were replaced with PubMed Unique Identifier (PMID). Details available at http://www.nlm.nih.gov/bsd/mms/medlineelements.html#pmid.

[9] http://www-tsujii.is.s.u-tokyo.ac.jp/~genia/corpus/GENIAontology.owl

MEDLINE:92043714

Charybdotoxin-sensitive, Ca(2+)-dependent membrane potential changes are not involved in human T or B cell activation and proliferation.

The involvement of ion channels in B and T lymphocyte activation is supported by many reports of changes in ion fluxes and membrane potential after mitogen binding. Human T and B lymphocytes demonstrate an early and transient hyperpolarization after ligand binding. Inasmuch as the change in membrane potential is dependent on elevation of free cytosolic calcium, the hyperpolarization is presumably through opening of Ca(2+)-stimulated K+ channels. We have used charybdotoxin, a known inhibitor of Ca(2+)-dependent K+ channels, to study the role of these channels in lymphocyte activation and mitogenesis. We demonstrate that charybdotoxin inhibits the ligand-induced transient membrane hyperpolarization in B and T cells in a dose-dependent fashion, without affecting changes in cytosolic Ca2+. However, blockade of the Ca(2+)-activated K+ channel is not associated with changes in cell-cycle gene activation, IL-2 production, IL-2R expression or B and T cell mitogenesis. These results imply that membrane potential changes secondary to the ligand-dependent opening of Ca(2+)-activated K+ channels are not involved in B and T lymphocyte activation and mitogenesis.

(a)

```
<article>
  <articleinfo>
    <bibliomisc>MEDLINE:92043714</bibliomisc>
  </articleinfo>
  <title>
    <sentence>
      Charybdotoxin-sensitive,
      <cons lex="Ca(2+)-dependent_membrane_potential_change" sem="G#other_name">
        <cons lex="Ca(2+)" sem="G#atom">Ca(2+)</cons>
        -dependent membrane potential changes
      </cons>
      are not involved in
      <cons lex="(AND (OR human_T_cell_activation human_B_cell_activation) (OR human_T_cell_proliferation
                 human_B_cell_proliferation))" sem="(AND (OR G#other_name G#other_name) (OR G#other_name G#other_name))">
        <cons lex="human*">human</cons>
        <cons lex="(OR *T_cell* *B_cell*)" sem="(OR G#cell_type G#cell_type)">
          <cons lex="T*">T</cons>
          or
          <cons lex="B*">B</cons>
          <cons lex="*cell">cell</cons>
        </cons>
        <cons lex="*activation">activation</cons>
        and
        <cons lex="*proliferation">proliferation</cons>
      </cons>
      .
    </sentence>
  </title>
  <abstract>
    <sentence>The involvement of
      <cons lex="ion_channel" sem="G#protein_N/A">ion channels</cons>
      in
      <cons lex="(AND B_lymphocyte_activation T_lymphocyte_activation)" sem="(AND G#other_name G#other_name)">
        <cons lex="B*">B</cons>
        and
        <cons lex="T*">T</cons>
        <cons lex="*lymphocyte_activation">lymphocyte activation</cons>
      </cons>
      is supported by many reports of changes in
      <cons lex="ion_flux" sem="G#other_name">ion fluxes</cons>
      and
      <cons lex="membrane_potential" sem="G#other_name">membrane potential</cons>
      after
      <cons lex="mitogen_binding" sem="G#other_name">
        <cons lex="mitogen" sem="G#protein_family_or_group">mitogen</cons>
        binding
      </cons>
      .
    </sentence>
    [...]
  </abstract>
</article>
```

(b)

Figure 2.4: Example of (a) a MEDLINE document (UI: 92043714, PMID: 1719077) and (b) its annotated version from the GENIA corpus. Indentation and extra vertical spaces are added to improve readability.

2.2.2 Evaluation measures and techniques

There are several quantitative, usually complementary, measures of the *effectiveness* of used methods, which denote the degree to which a hypothesis reflects the target function (in the case of classification) or to which the obtained results are correct and complete (in the case of information retrieval). The most popular, used in both classification and information retrieval, are *precision* and *recall*.

In information retrieval, precision P is seen as the proportion of retrieved elements (e.g., documents, relations) that are relevant to the total number of the *retrieved* elements, whereas recall R is the proportion of retrieved and relevant elements to the total number of the *relevant* items, i.e.,

$$P = \frac{|retrieved \cap relevant|}{|retrieved|}, \qquad R = \frac{|retrieved \cap relevant|}{|relevant|}. \tag{2.3}$$

In text categorization, precision and recall are calculated based on the *contingency matrix* shown in Figure 2.5.

		Actual label	
		positive	*negative*
Predicted label	*positive*	TP	FP
	negative	FN	TN

TP - true positive, FP - false positive,
FN - false negative, TN - true negative

Figure 2.5: Contingency matrix

True positives (TP) and *true negatives* (TN) are the numbers of elements *correctly* labeled as belonging or not belonging, respectively, to the given class. *False positives* (FP) and *false negatives* (FN) are the numbers of elements *incorrectly* labeled as belonging or not belonging, respectively, to the given class. Precision is then seen as the ratio of true positives by the total number of elements labeled as positive, whereas recall is the ratio of true positives by the total number of elements that should be labeled as positive, i.e.,

$$P = \frac{TP}{TP + FP}, \qquad R = \frac{TP}{TP + FN}. \tag{2.4}$$

In real-world scenarios the goal is to balance precision and recall since the two negatively influence each other, i.e., setting the parameters of a classification or information extraction method to increase precision may result in decreasing recall, and vice versa. The most often used measure that shows this balance is the F_1 measure [146]:

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}. \tag{2.5}$$

In the case of multi-class classification, there is a necessity of combining single results from contingency matrices built for each class. There are two different ways of averaging the obtained results [131]:

- *macro-averaging*, which is an arithmetical average of measures calculated for each class individually, and
- *micro-averaging*, which is an average calculated by combining TP, TN, FP, and FN across all classes into a single contingency matrix.

Macro-averaging reflects well the performance of a classification system with unevenly distributed classes, whereas micro-averaging favors larger classes in expense of poorer results for small ones.

While evaluating a classifier, a set of pre-categorized elements is usually divided into three parts:

- a *training* set based on which the classifier is built (the hypothesis is created),
- a *validation* set based on which the parameters of the classifier are tuned, and
- a *testing* set based on which the classifier is evaluated.

Validation (using the validation set) is the process of evaluating the classifier's parameters that have been used in the *learning* process (using the training set). *Testing* is the process of evaluating the classifier after all the parameters have been optimized. The validation and test sets should always be disjointed to avoid *evaluation bias*.

2.3 Related work

This section briefly discusses related work revolving around classification and relation extraction, focused mainly on the biomedical domain. Further discussion of work related specifically to the methods used in Chapters 3 to 5 are provided in the respective chapters.

The OHSUMED corpus as a subset of the MEDLINE database has been used by many researchers to perform classification using MeSH terms as class labels. Due to the huge number of categories (MeSH terms), most of the investigations have been focused on smaller subsets of the MeSH thesaurus. For instance, the MeSH tree structure was reduced to a particular subtree, such as *Heart Diseases*, and the documents were classified using various learning techniques such as k-nearest neighbours (kNN) [86, 150], linear classifiers [92, 150], or Neural Networks (NN) [127]. Although the Heart Diseases subtree represents

the multi-label problem, the distribution of categories shows that vast majority of them are assigned to one document only.

Other reductions include limiting the category pool to those that occur more than 75 times in the OHSUMED dataset [87]. The authors used instance-based learning to assign documents to the selected MeSH terms. The Naïve Bayes (NB) classifier was used in the investigation of optimal training sets for classification of MEDLINE documents to a small set of 20 MeSH terms [135]. The optimal sets were searched for each MeSH term separately and involved building the sets by including the documents that were assigned to a given term but excluded the documents that were assigned to terms "closest" to the one for which the set was being built. The NB classifier was also used to classify MEDLINE documents to Gene Ontology (GO) [78].

Other approaches involve a combination of machine learning methods, linguistic parsing, and lexicons. MetaMap [18] is a linguistic and lexicon-based approach that processes text by chunking it to noun phrases and generating lexical variants, which are further compared against UMLS Metathesaurus [9]. MetaMap is also a major component of NLM's Medical Text Indexer (MTI) [19]. A large number of categories, almost 20,000, was reported in [126] with three methods based on regular expression matching, vector space model, and the combination of the two. The advantages of these approaches lie in virtually no learning (model building) process. However, the evaluation was performed on a small set of 1,000 documents, and the author aimed to maximize the precision of the methods, neglecting recall.

The comparative evaluation of MetaMap [18], MTI [19], and the approach presented in [126] was shown by Trieschnigg *et al.* [144], who attempted categorization of 1,000 MEDLINE articles from year 2008 to the MeSH terms, focusing on maximizing the precision of the methods (similarly to [126]). The authors additionally evaluated two concept-oriented methods, which rely on creating probabilistic models for each of the MeSH terms by combining all the documents assigned to these terms, as well as another instant-based method, kNN, which proved to be the most effective.

As opposed to document-oriented classification, relation extraction is sentence-oriented, and therefore involves smaller sets of documents than those used in classification. Extracting entity relations have been attempted using (1) predefined sets of patterns with the output of a shallow phrase structure parser, (2) sets of logic rules or spanning graph heuristics applied on the output of a dependency parser, (3) combination of dependency parsing with machine learning, and (4) rarely the output of a constituent tree. An example of processing sentences with a shallow phrase parser and a set of flat patterns was shown in [37]. The authors used GENIA as an input corpus and two simple patterns to extract relations consisting of a verb or a verb followed by a preposition. Pro3Gres [129], a typed dependency parser,

was used to create a set of Prolog rules to combine several dependencies to form relations between entities [121, 123, 122]. The authors evaluated the performance of the linguistic performance on a subset of GENIA and another 147-sentence set of annotated MEDLINE abstracts. During evaluation they considered only a few verbs and were interested in the biological significance of the obtained results. A similar dependency-like approach was used in [36]. The authors used the constituent parser to built dependency graphs between the tree constituents. They looked for entity relations by traversing the graph with a handful of heuristics and constraints. Similarly to [37] they were interested in generalizing the relations to an appropriate GENIA ontology level. BIEQA [13, 14] combines linguistic analysis and co-occurrence-based principles to extract relations present in GENIA. It also uses co-occurrence to find *feasible* relations between the annotated terms. BioPatentMiner [101] identifies biological terms and relations from patent databases and integrates the obtained information into biomedical ontologies. The system uses BioAnnotator [141], a tool for dictionary-based identification and classification of biological terms that refer to UMLS Metathesaurus [9] as a dictionary. Relations between the terms are searched using a template with a set of predefined verbs.

A combination of dependency parsing and machine learning techniques has also been investigated [30, 29, 21, 22]; however, although these attempts usually do not require any manual work, they exhibit inferior effectiveness when compared to the methods that require the input of an expert user to prepare a set of patterns or rules.

A number of investigations have been undertaken in the study of protein-protein interactions (PPI) [69, 46, 45, 100, 152, 35], which is considered a specific case of relation extraction. The PPI extraction has the advantage of knowing the verbs and verb expressions indicating interaction beforehand.

Chapter 3

Multi-label Associative Classification

3.1 Introduction

Multi-label classification is the process of categorizing objects (in the case of this dissertation, MEDLINE documents) to one or more classes (labels). The process of building classification models (learning) can be realized with a variety of machine learning techniques such as Support Vector Machines (SVM), Naïve Bayes (NB), k-Nearest Neighbours (kNN), decision trees, etc., some of which need to undergo sophisticated adaptation to suit *multi-label* classification. This work addresses *associative classification*, a relatively new classification method, which has been recently gaining researchers' attention [95, 93, 154, 155, 23, 142]. The advantage of using associative classification lies in 1) the simplicity of the core idea, i.e., using a statistical approach (finding frequent patterns), which may be perceived as conceptually simpler and more intuitive than creating a mathematical model (such as in SVM or NB classifiers), and 2) the descriptive nature of the model, i.e., a flat list of human-interpretable and independent (modular) rules. Therefore, not only do the rules possess predictive power, but they can also be utilized in other applications such as highlighting the most important words, i.e., the words that appear on the left-hand side of the rule, in documents or document summarization. Additionally, the rules can be presented to a user who could manually adjust and remove them, and even add new rules, to improve the prediction model.

It has been shown [33] that associative classification is more accurate in text classification when compared to other descriptive classifiers. Moreover, it has been argued [144] that binary classifiers, such as the very effective SVM, require considerable amount of effort to be adapted to multi-label classification and are not suitable for tasks with large number of classes. This is the case with MeSH, which is considered in this work.

Although multi-label classification has been widely studied [90, 31, 96, 97, 57], a relatively

small amount of work has been devoted to multi-label *associative* classification. Several different associative classifiers, such as CBA [95], CMAR [93], CPAR [154], ARC-AC/BC [155], or L_3^M [23] have been proposed. However, most of them consider only single-label classification. As an exception, ARC-AC/BC [155] uses *dominance factor*, a score proportional to the frequency of classes appearing in rules matching the document, in combination with the variation of SCut thresholding to determine a subset of classes that should be assigned to the document. MCAR [143] and MMAC [142] consider multiple labels, though the generated association rules are built by iteratively repeating the method for generating single-label rules. Furthermore, more effort has been dedicated to investigating different classification schemata based on previously generated association rules rather than the efficient generation of these rules, which is justified providing that the above methods were designed to work with small datasets. Another considerable limitation of these methods is that they do not handle observations with repeated features, i.e., they do not account for recurrence of words in documents, and instead acknowledge only the binary presence or absence of words. However, it has been recognized that the repetition of words is significant, hence the common use of TF/IDF (i.e., the frequency of a word in a document relative to the frequency of the word in a collection) in the vector-space representation of documents [73, 124].

As opposed to the aforementioned efforts, the proposed approach aims at *multi-label* associative classification with *recurrent items*, hereinafter referred to as *MLACRI*, with a special emphasis on *self-sufficient* models, i.e., models that do not require any further adaptation to multi-label classification, that can be applied to *large datasets* (containing hundreds of thousands of documents).

The process of discovering frequent patterns, the first step towards forming association rules (see Section 2.1.1), is undoubtedly the most demanding in terms of computational complexity. The maximum number of patterns over the alphabet of n items (the number of words in a collection) is 2^n out of which only a small portion may be *frequent*, i.e., satisfying the support threshold. In real-life applications, such as classification of MEDLINE documents to MeSH, it is not unusual that n represents values in several thousands. Verifying all the possible patterns against the dataset of, again, often hundreds of thousands of documents may be an unfeasible task. Numerous algorithms have been developed so far to address this problem. Two main methodologies include *apriori-based* and *pattern-growth-based* approaches. *Apriori* [16] (and its variations) is an algorithm that utilizes so called *apriori property* and states that if an itemset is not frequent (does not meet the support threshold), its supersets cannot be frequent either. Beginning with atomic itemsets, which consist of one item only, the algorithm extends them while discarding these itemsets that are not frequent, which reduces the number of generated patterns that would never be frequent. Later improvements of the Apriori algorithm include a *tree projection* algorithm [15], which sig-

nificantly reduces the number of itemset candidate tests. A new, even more efficient version of tree projection is the key component of MLACRI.

The remainder of this chapter is organized as follows: Section 3.2 presents work related to associative classification, which is followed by a formal problem definition in Section 3.3. The inner-workings of generating multi-label, associative-classification, recurrent-item rules are described in Section 3.4, whereas Section 3.5 discusses different classification strategies. Quantitative and qualitative evaluations of MLACRI are given in Section 3.6.

A version of this chapter has been published in parts in [119, 113, 114, 116].

3.2 Related work

Integration of association rule mining with classification was originally shown in the CBA algorithm [95]. The authors extended the commonly used Apriori algorithm [16] to generate classification rules. Similar approach has been employed in the family of associative classification algorithms ARC-AC/BC [155]. However, Apriori-based algorithms, that use *candidate set generate-and-test* approach are computationally expensive, especially with long and numerous patterns in an input dataset. As an alternative, the pattern-growth family of algorithms has been proposed [63]. This approach adopts a method to project and partition the dataset based on the currently discovered patterns. Its first implementation, *FP-growth* [65, 66], is based on a *frequent-pattern tree* (FP-tree) and was claimed to be an order of magnitude faster than the Apriori algorithm. *FP-growth* has been used in associative classification in CMAR [93].

Another approach to accelerate rule generation and improve classification accuracy was proposed in CPAR [154], an algorithm that integrates rule-based methods, such as FOIL/FFOIL [109, 111, 112, 110] and RIPPER [40, 41], to generate rules with features of associative classification in predictive rule analysis. A technique based on *intersection method* [156] has been proposed in MCAR [143] and MMAC [142].

Currently there are several techniques that perform efficient projection of the data to generate association rules. A pattern-growth approach [63] adopts a *divide-and-conquer* method to project and partition the dataset based on the currently discovered patterns. This method has been applied in, e.g., FP-growth [65], FreeSpan [64], PrefixSpan [105], and as a framework for parallel data mining [44]. A similar divide-and-conquer approach has been applied to the *tree projection* algorithm [15], where frequent itemsets and their projected datasets are embedded in a tree structure. Although the data-projection-based technique can be applied to both Apriori and pattern-growth families of algorithms, the introduction of recurrent items appeared to be a challenge for the latter. For instance, the only FP-tree-based

approach known to incorporate recurrent items [104] is not capable of discovering *all* frequent patterns in text. The Apriori algorithm is more flexible in this matter as it does not impose restrictions on how data is stored, which seems to be the major issue with FP-tree.

The core component of MLACRI, i.e., discovering frequent patterns (itemsets), is similar to the tree projection algorithm described in [15] in that the information about itemsets together with their projected datasets are organized in a tree structure. Main differences are that the nodes of the MLACRI tree represent items instead of the whole itemsets, and that new nodes are created directly from the tree without using additional structures (which require additional space) such as triangular matrices used in [15]. Such a structure allows for a significant reduction of the number of candidate tests, which is a crucial problem for association-rule algorithms, which produce candidates to obtain longer itemsets. Furthermore, MLACRI uses less space to store the projected datasets [116]. In order to accommodate the algorithm dealing with class labels (or more precisely, with multiple class labels) as well as with recurrence of items in a single transaction, further modifications have been imposed on the projected tree.

3.3 Problem definition

Let C be a set of labels and $\mathcal{I}$ a set of items. The dataset D consists of transactions[1] being a powerset of C and $\mathcal{I}$ in the form of $\langle X_i, C_i \rangle$ where $X_i \subset \mathcal{I}$ is a set of items $\{x_{i1}, x_{i2}, \ldots, x_{ik}\}$ and $C_i \subset C$ is a set of labels $\{c_{i1}, c_{i2}, \ldots, c_{ij}\}$ for each transaction T_i with j labels and k items, such that $\bigcup_{i=1}^{n} T_i = D$.

The task of associative-classification rule generation is to find association rules in the form of $X \rightarrow C$ indicating a strong relationship between items in X and the set of classes C. (Traditional associative classification considers C a single class as opposed to a multi-label scenario presented in this chapter.) The set of items X in a rule is commonly called a *condition set* or simply *condset*.

There are two measures indicating the strength of a rule. The *support* σ of the rule $X \rightarrow C$ is the fraction of transactions in D that contain both X and C, or formally:

$$\sigma_D(\langle X, C \rangle) = \frac{\hat{\sigma}_D(\langle X, C \rangle)}{|D|} \tag{3.1}$$

where $\hat{\sigma}_D(x)$ denotes the number of occurrences of x in D.

The *confidence* φ of the rule is the fraction of transactions containing X which also contain

[1] The remainder of this chapter refers to documents and words as transaction and items, respectively (see Section 2.1.1 for explanation).

C, or formally:

$$\varphi_D(\langle X, C\rangle) = \frac{\hat{\sigma}_D(\langle X, C\rangle)}{\hat{\sigma}_D(X)} \tag{3.2}$$

In *recurrent-item* associative classification transactions are in the form of $\langle\{\rho_1 x_1, ..., \rho_n x_n\}, C\rangle$, where $x_i \in \mathcal{I}$ is an item, $C \subset \mathcal{C}$ is a set of labels, and ρ_i is the number of occurrences of the item x_i in the transaction.

Let $T = \langle X_T, C_T\rangle$ be a transaction such that $X_T = \{\rho_{1T} x_1, \ldots, \rho_{mT} x_m\}$ and $C_T = \{c_1, \ldots, c_n\}$, and $R = \langle X_R, C_R\rangle$ be a ruleitem such that $X_R = \{\rho_{1R} x_1, \ldots, \rho_{kR} x_k\}$ and $C_R = \{c_1, \ldots, c_l\}$. Transaction T *supports* ruleitem R if $\forall i \in [1, l] : c_i \in C_R \rightarrow c_i \in C_T$ and $\forall j \in [1, k] : x_j \in X_R \rightarrow x_j \in X_T \wedge \rho_{jR} \leq \rho_{jT}$. Less formally, a transaction supports a ruleitem if each item and label from the ruleitem has its counterpart in the transaction and the number of occurrences of each corresponding item in the transaction is no less that this in the ruleitem.

A simple example enhancing the difference between recurrent- and non-recurrent-item representation is shown in Figure 3.1.

uncategorized document D: $\{a, b, c, d, a, b, c, a, c, a, c\}$

non-recurrent-item representation	*recurrent-item* representation
$D = \{a, b, c, d\}$	$D = \{4a, 2b, 4c, 1d\}$
rules	rules
$R_1 = \langle\{a, b, d\}, \{C_1, C_2\}\rangle$	$R_1 = \langle\{3a, 2b, 1d\}, \{C_1, C_2\}\rangle$
$R_2 = \langle\{a, b, c\}, \{C_2, C_3\}\rangle$	$R_2 = \langle\{3a, 3b, 2c\}, \{C_2, C_3\}\rangle$
Both R_1 and R_2 match D	Only R_1 matches D ($\{3a, 3b, 2c\} \not\subset D$)

Figure 3.1: Difference between *non-recurrent-* and *recurrent*-item representation

Recurrent-item representation allows for further discrimination of the rules based on the number of recurrent items in both the document and rules. In the given example, both rules R_1 and R_2 in non-recurrent-item representation match document D, however, when the recurrence of items is considered, R_2 no longer matches D due to an excess amount of item b.

3.4 Generating multi-label, recurrent-item, classification rules

This section discusses the design of the proposed algorithm and, to ease reading, is broken down into several parts, each introducing a new concept. Section 3.4.1 describes the problem of generating frequent patterns and although it does not consider class labels, it is an essential step in generating associative-classification rules. Sections 3.4.2 to 3.4.5 discuss the extensions, constraints, and optimization techniques applied to the basic frequent pattern generation in order to generate multi-label, recurrent-item, associative-classification

rules. The final algorithm comes in two flavors that are based on breadth-first and depth-first search algorithms as presented in Section 3.4.6.

3.4.1 Frequent pattern mining

The item enumeration is based on a tree, where nodes represent items and labels and paths from the root of the tree to terminal nodes are graphical representations of rules.

To improve the transparency of the description we initially describe the problem without considering class labels, focusing on items only. The tree discussed in this section is therefore called an *itemset tree* in contrast to a *ruleitem tree*, the details of which are discussed in Section 3.4.2.

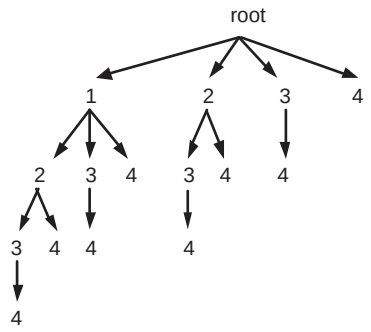
Following the problem defined in section 4.3, it is further assumed that there is an order between the items in each transaction, e.g., based on the position of the items in the input dataset. Expression $x_i < x_j$ denotes that item x_i precedes x_j. The itemset tree is defined as follows:

1. Each node (vertex), except the root node, in the tree represents an item in $\mathcal{I}$.
2. Each edge corresponds to an order between two items in a transaction.
3. The root node does not correspond to any item and does not have any incoming edges.
4. Each path of length l in the tree connecting the root node with any other node corresponds to an l-itemset, either frequent or hypothetical (candidate), such that each node in the path represents a single item in the itemset.

An example of an itemset tree and the corresponding set of itemsets is shown in Figure 3.2.

The tree in this figure is *complete*, i.e., it consists of all possible itemsets that can be created from four items. The nodes are *generated* starting with the enumeration of the items at the first level (the closest to the root node). The following levels are generated by rewriting the items in the nodes following (standing to the right of) each of the current-level nodes. The itemsets shown in this example correspond to paths spanning from the root of the tree to its leaves. For instance, the longest itemset $\{1, 2, 3, 4\}$ is represented by the far-left path spanning from the root, through nodes 1, 2, and 3 to node 4.

For simplicity, given a node p and an item x represented by p, p is referred to as if it were actually the item x. For any two nodes p and q, the expression $p = q$ denotes that both represent the same item.



(a)

1-itemsets: $\{1\}, \{2\}, \{3\}, \{4\}$
2-itemsets: $\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}$
3-itemsets: $\{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}$
4-itemset: $\{1,2,3,4\}$

(b)

Figure 3.2: Example of (a) an itemset tree and (b) its corresponding itemsets

Definition 3.4.1 (Candidate itemset and frequent itemset) *A k-itemset, i.e., a set of k items, is* frequent *if it satisfies a support threshold ξ. A $k+1$-itemset is a* candidate $k+1$-itemset *if it was obtained by adding an item to a frequent k-itemset.*

Definition 3.4.2 (Candidate node and frequent node) *A node q in an itemset tree is a* candidate node *if all of its ancestor nodes form a frequent itemset. The node q is* frequent *if it is one of the nodes forming a frequent itemset.*

The intuition behind candidate and frequent itemsets is that a candidate itemset becomes frequent if it satisfies a support threshold of ξ. A similar analogy applies to the candidate and frequent nodes. It is important to note that according to the above definitions, a candidate node can only be generated from a frequent node, which is the very essence of *Apriori property*, i.e., if an itemset is not frequent, its superset cannot be frequent either, and therefore any further generation of candidate nodes based on this itemset is of no avail.

The outcome of an algorithm for generation of frequent itemsets is the itemset tree consisting of frequent nodes only. The approach taken here is similar to the one in the Apriori algorithm [16] in that it produces candidate nodes that are tested against a set of transactions. However, in the proposed algorithm candidate itemsets are tested only against a subset of transactions, which narrows down with an increasing length of itemsets. This reduction in the number of itemset–transaction comparisons is facilitated through *projected datasets*.

Definition 3.4.3 (Projected dataset) *Given a node q, a dataset $D_q \in D$ is a* q-projected dataset *if for each transaction $T_i \in D_q$, $q \in T_i$. $D_q = D$ if $q = r$, where r is the root node.*

Definitions 3.4.2 and 3.4.3 imply that $D_q \subset D_p$ if p is the parent of q. This is an important property as it guarantees that a projected dataset at some node is no bigger than the projected dataset of this node's parent.

The pseudocode of the general behavior of the algorithm with respect to generation of frequent itemsets is shown in Figures 3.3 to 3.5, whereas Table 3.1 contains the symbols and terminology used in the algorithm and the following sections.

Table 3.1: Symbols used in rule generation

$\sigma, \hat{\sigma}$	Support and support count, respectively
$\xi, \hat{\xi}$	Support threshold and support threshold count, respectively
ρ	Number of word occurrences
`Parent(`q`)`	Parent of node q
V_l	Set of nodes at level l
C_p	Set of candidate nodes that are children of node p
F_p	Set of frequent nodes that are children of node p
S_q	Set of siblings that follow (stand to the right of) node q
D_q	Set of transactions at node q (q-projected dataset)

input : root node r
set of items $\mathcal{I}$
support count threshold $\hat{\xi}$
output: set of frequent nodes F

```
1  C_r := I
2  CountSupport(C_r)
3  F_r := Prune(C_r, ξ̂)
4  V_1 := F_r
5  l := 1
6  while V_l ≠ ∅ do
7      for each p ∈ V_l do
8          C_p := GenerateCandidates(S_p)
9          CountSupport(C_p)
10         F_p := Prune(C_p, ξ̂)
11         V_{l+1} := V_{l+1} ∪ F_p
12     l = l + 1
```

Figure 3.3: Pseudocode of the general algorithm for generation of frequent itemsets

Given a set of all possible items $\mathcal{I}$ in dataset D, the first level of nodes is generated based upon the frequency of the occurrences of these items in the dataset (Figure 3.3). Function `CountSupport` traverses the dataset and increases the support count for each node from

the set passed to the function as an argument. Infrequent nodes are pruned based on the support threshold. Subsequent levels are created based on preceding levels. The three functions, `GenerateCandidates`, `CountSupport`, and `Prune` are repeated for each node at the current level. The algorithm stops if there is no new level of nodes to generate further nodes.

```
for each q ∈ C_p do
    for each T ∈ D_p do
        if q ∈ T then
            σ̂(q) := σ̂(q) + 1
            D_q := D_q ∪ T
```

Figure 3.4: Pseudocode of the `CountSupport` function

For a given node p the `GenerateCandidates` function simply creates copies of p's siblings and adds them as p's own children. Function `CountSupport`, shown in Figure 3.4, verifies the existence of each generated candidate against a projected dataset by testing if the candidate exists in the transactions of this dataset. If it does exist, the candidate's support count is increased by one, whereas T is added to the candidate's projected dataset[2]. Function `Prune`, shown in Figure 3.5, simply verifies the support count of each candidate and prunes those that do not satisfy the given support threshold.

```
F_p := C_p
for each q ∈ C_p do
    if σ̂(q) < ξ̂ then
        F_p := F_p \ q
```

Figure 3.5: Pseudocode of the `Prune` function

3.4.2 Class labels in the projected tree

Although probably the most obvious way of including labels in an itemset tree is to treat the labels as items, i.e., to neglect the distinction between items and labels, such a solution would result in a vast number of association rules without labels, as well as rules consisting of labels only. In other words, there would be cases where creating a rule in the form of $X \rightarrow C$ would be impossible due to the lack of either X or C. Although after pruning incomplete rules this solution is undoubtedly correct, it is highly inefficient. For instance, in the tree shown in Figure 3.2, if "1" denotes a label and "2", "3", and "4" denote items, from the total of 15 rules only seven would have both a label and at least one item.

[2]The actual implementation counts the support and creates projected datasets using more advanced techniques. More specifically, transactions are verified against the entire set of candidates C_p *at once* and projected datasets keep only references to transactions in dataset D. A detailed discussion is given in Section 3.4.4.

The proposed modification of the tree to efficiently enumerate association rules with class labels includes the following constraints:

- The first level consists of nodes that represent *labels* only.
- The second level consists of nodes that represent *items* only.
- Level three and higher consist of nodes that represent both items and labels.
- If nodes have the same parent, nodes representing items precede those representing labels.

An example of a complete tree consisting of items and labels with the aforementioned constraints is depicted in Figure 3.6.

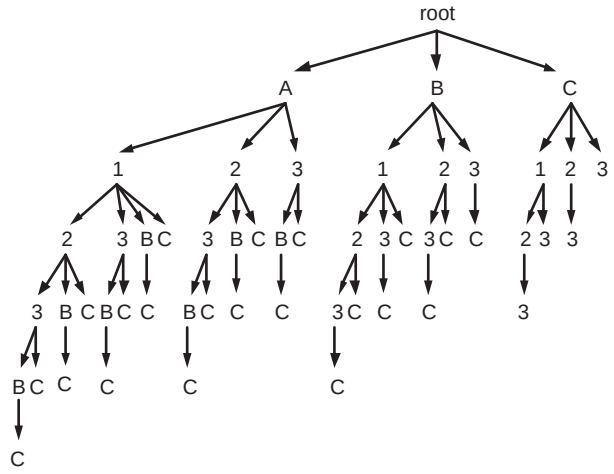


Figure 3.6: Example of a multi-label ruleitem tree. Labels and items are denoted by capital letters and numbers, respectively.

Placing labels at the first level ensures that all rules have at least one label. A lack of labels at the second level prevents generating rules without items. It is important to note that though the first level of labels is essential for building further, larger itemsets, it cannot be used to produce any rules by itself.

When relating the new constraints to the basic algorithm, it is observed that the second level candidates cannot be generated from S_p of any first-level node p. Similarly the third level of nodes does not fully follow the `CandidateGeneration` function. However, the generation of the nodes for levels four and higher fully complies with this function. This leads to the modifications of the basic algorithm as shown in Figure 3.7.

The new algorithm consists of four parts. The first part builds the first level of nodes representing labels (lines 1–4). The second part (lines 5–9) adds items as children to each first-level node. At this point the fully qualified rules $X \rightarrow C$ consist of exactly one item and one

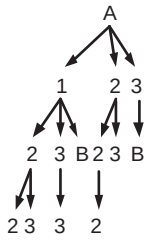
```
   input  : root node r
            set of labels C
            set of items I
            support count threshold ξ̂
   output: set of frequent nodes F
 1 C_r := C
 2 CountSupport(C_r)
 3 F_r := Prune(C_r, ξ̂)
 4 V_1 := F_r
 5 for each p ∈ V(1) do
 6 |   C_p := I
 7 |   CountSupport(C_p)
 8 |   F_p := Prune(C_p, ξ̂)
 9 |   V_2 := V_2 ∪ F_p
10 for each p ∈ V_2 do
11 |   C_p := GenerateCandidates(S_p ∪ S_Parent(p))
12 |   CountSupport(C_p)
13 |   F_p := Prune(C_p, ξ̂)
14 |   V_2 := V_2 ∪ F_p
15 l := 3
16 while V_l ≠ ∅ do
17 |   for each p ∈ V_l do
18 |   |   C_p := GenerateCandidates(S_p)
19 |   |   CountSupport(C_p)
20 |   |   F_p := Prune(C_p, ξ̂)
21 |   |   V_{l+1} := V_{l+1} ∪ F_p
22 |   l = l + 1
```

Figure 3.7: Pseudocode of a general algorithm for mining a ruleitem tree

label can be produced. Unlike the first two levels, the generation of the third level involves the two preceding levels (lines 10–14). Given node p at level two, the third-level candidates are generated based on p's siblings S_p as well as siblings of p's parent $S_{\texttt{Parent}(p)}$. Level four and higher are generated in the same fashion as described in the previous section (lines 15–22).

3.4.3 Recurrent items in the projected tree

A fragment of a tree consisting of recurrent items and its corresponding set of ruleitems is shown in Figure 3.8. To avoid ambiguity between item's identifier and a number of its occurrences, the latter is put in round brackets, i.e., notations ρx_i and $x_i(\rho_i)$ are equivalent.



(a)

2-ruleitems:	$\langle\{1(1)\},\{A\}\rangle,\ \langle\{2(1)\},\{A\}\rangle,\ \langle\{3(1)\},\{A\}\rangle$
3-ruleitems:	$\langle\{1(1),2(1)\},\{A\}\rangle,\ \langle\{1(1),3(1)\},\{A\}\rangle,$
	$\langle\{1(1)\},\{A,B\}\rangle,\ \langle\{2(2)\},\{A\}\rangle,$
	$\langle\{2(1),3(1)\},\{A\}\rangle,\ \langle\{3(1)\},\{A,B\}\rangle$
4-ruleitems:	$\langle\{1(1),2(2)\},\{A\}\rangle,\ \langle\{1(1),2(1),3(1)\},\{A\}\rangle,$
	$\langle\{1(1),3(2)\},\{A\}\rangle,\ \langle\{2(3)\},\{A\}\rangle$

(b)

Figure 3.8: Fragment of (a) a tree with *recurrent items* and (b) its corresponding set of ruleitems

The adaptation of the algorithm to account for recurrence of items requires a simple modification of the definition of the set of siblings S. Given a node p, the new set of sibling S_p is augmented by p itself if

- $p \in \mathcal{I}$, and
- the number of p's ancestors representing the same item in the path from the root node to p is less that the maximum number of occurrences of p in any of the transactions.

The first constraint comes from the fact that labels in a rule must be distinct, whereas the latter ensures that the maximum number of occurrences of any item in a ruleitem is no larger than the maximum number of occurrences of that item in any transaction.

3.4.4 Candidate test optimization

Most of the existing solutions related to the generate-and-test approach in frequent itemset mining rely on a simple candidate test method. Each candidate itemset is verified against each transaction from a projected (or original) dataset.

Let $X = \{x_1, x_2, \ldots, x_k\}$ be a candidate k-itemset and $T = \{y_1, y_2, \ldots, y_l\}$ be a transaction of length l. The easiest strategy to test the candidate is to compare each item in X with every item in T. In the worst case scenario this results in $k \times l$ comparisons. If items in both X and T are ordered (the order in X is embedded into the generated tree, whereas the order in T requires a single sort operation) then the complexity decreases to the size of the transaction. This, however, has to be repeated $|D_p|$ times for every node p in the tree.

A new method of ruleitem-tree-based candidate testing is proposed as follows. It is observed that (1) maintaining projected datasets allows for checking only the last item in an itemset, which is represented by a node in C_p for some p (since the remaining part of the itemset has already been tested in the previous levels), and (2) C_p is ordered in the tree. Therefore, the entire C_p can be tested against a transaction *all at once*. Instead of testing whether $X \subset T$, the support of each item $x_i \in X$ is increased if $x_i = y_j$ for any $y_j \in T$. Thus, the new approach results in $|C_{\texttt{Parent}(p)}|$ times fewer comparisons.

Function `IncreaseSupportAndCreateProjDatasets`, shown in Figure 3.9, performs the candidate test described above.

```
input : transaction T
        set of candidates C_p
1  x := Next(T)
2  q := Next(C_p)
3  while x ∉ ∅ ∧ q ∉ ∅ do
4      if x = q ∧ ρ(x) ≥ ρ(q) then
5          σ̂(q) := σ̂(q) + 1
6          D(q) := D(q) ∪ T
7      else
8          if x > q then
9              q := Next(C_p)
10         else
11             x := Next(T)
```

Figure 3.9: Pseudocode of the `IncreaseSupportAndCreateProjDatasets` function

Function `Next`(X) successively returns items from set X, one at each call, with respect to the order in X.

3.4.5 Dataset optimization

The algorithm distinguishes between two types of datasets: *generic* and *projected*. The generic dataset is read from a hard drive and stored in main memory. This dataset is kept at the root node of the ruleitem tree and used for a candidate test at the first level of the tree. The projected datasets are obtained from the generic dataset and used for the candidate test at levels higher than one. This section discusses the optimization techniques for storage and accessing both generic and projected datasets.

One possibility of storing a dataset in the main memory is to keep transactions in the form of bit vectors such that each bit represents either the existence or absence of an item. This allows for a very fast candidate test if itemsets are kept in the same fashion. However, this solution is memory consuming, especially when dealing with sparse data which is almost always the case in text categorization (due to a large vocabulary). Another solution is to keep items in a transaction in the form of a list. Although it slows down candidate tests, this structure is very often the only way to deal with large amount of sparse and highly dimensional data, and therefore is used in MLACRI.

To reduce the amount of required memory, the stored transactions consist of frequent items only. Frequent items can be filtered out by either 1) reading the complete transactions once into the main memory and then pruning infrequent items, or 2) reading the complete transactions to compute item frequency values and then reading the data once again and storing in the main memory only the frequent items, selected based on the counts computed during the first reading. The former solution is faster as it reads the dataset only once, whereas the latter is indispensable when there is not enough space to load the entire dataset. MLACRI implements both by giving a user the option to choose from the two.

Once the entire (generic) dataset is in the main memory, the *projected* datasets store only references to particular transactions in this dataset, which additionally reduces memory consumption.

Another optimization employed in MLACRI involves deleting projected datasets when they are no longer needed. For instance, once the level $k + 1$ has been created, the projected datasets of nodes at level k are deleted.

3.4.6 The proposed multi-label, recurrent-item, rule generation algorithm

This section discusses two strategies for exploring the ruleitem tree, the breadth-first search and the depth-first search. These strategies differ from each other with respect to the total number of projected datasets required to compute the rules and the time needed to generate the tree.

Breadth-first strategy

In a breadth-first search approach the tree is explored level by level, i.e., level $l + 1$ of the tree is generated only if computations are completed for all nodes at level l. The basic mechanism of the breadth-first search is employed in the algorithm presented in the previous sections. Here more details are provided with a discussion of pros and cons of this search.

As opposed to the depth-first search, the breadth-first search takes advantage of Apriori property, i.e., if an itemset is not frequent its superset cannot be frequent either. Due to the fact that the tree is a representation of itemsets, this property can be exploited to optimize the algorithm by introducing the concept of *base node*.

Definition 3.4.4 (Base node) *Given nodes p and q such that $q \in C_p$, a node $s \in S_p$ is a* base node of *q, denoted* `Base(q)`*, if q has been directly generated from node s.*

Before generating an $(l + 1)$-level node q for the l-level node p from some $s \in S_p$, the algorithm verifies if s belongs to frequent nodes of p's $(l - 1)$-level base node, i.e. whether $s \in F_{\texttt{Base}(p)}$. This prevents testing nodes that are guaranteed to not be frequent, and thus saves time needed for candidate tests. This new procedure, `GenerateCandidatesBF`, is depicted in Figure 3.10, whereas Figure 3.11 presents the complete breadth-first search algorithm.

input : node p
set of sibling S_p
set of frequent nodes F
output: set of candidates C_p
1 **for** *each* $s \in S_p$ **do**
2 **if** $s \in F_{\texttt{Base}(p)}$ **then**
3 $C_p \coloneqq C_p \cup s$

Figure 3.10: Pseudocode of the `GenerateCandidatesBF` function

The algorithm begins with determining a set of first-level frequent nodes. After employing dataset optimization, as described in section 3.4.5, this set is equal to the set of labels C. A set of candidates C_p is generated for each node in the current level l (lines 8–13) following the corresponding procedures related to level l (discussed in Section 3.4.2). After determining the frequency of candidates in the transactions and creating projected datasets (lines 14–15), the infrequent candidates are pruned (line 16) and the current projected dataset is removed from the memory (line 18). The procedure is repeated as long as new frequent nodes are generated.

```
input  : root node r
         set of labels C
         set of items I
         support count threshold ξ̂
output: set of frequent nodes F
1  C_r := C                                                    ◄ 1st level
2  CountSupport(C_r)
3  F_r := Prune(C_r, ξ̂)
4  V_1 := F_r
5  l := 1
6  while V_l ≠ ∅ do
7      for each p ∈ V_l do
8          if Parent(p) ∈ ∅ then
9              C_p := I                                        ◄ 2nd level
10         else if Parent(p) ∉ ∅ ∧ Parent(Parent(p)) ∈ ∅ then
11             C_p := GenerateCandidatesBF(S_p ∪ S_Parent(p))  ◄ 3rd level
12         else
13             C_p := GenerateCandidatesBF(S_p)                ◄ levels 4 and up
14         for each T ∈ D_p do
15             IncreaseSupportAndCreateProjDatasets(T, C_p)
16         F_p := Prune(C_p, ξ̂)
17         V_{l+1} := V_{l+1} ∪ F_p
18         Delete(D_p)
19     l = l + 1
```

Figure 3.11: Pseudocode of the breadth-first algorithm for multi-label, recurrent-item, associative-classification rule generation

Depth-first strategy

The main drawback of the breadth-first algorithm is that the number of projected datasets is equal to the number of nodes in the current level l and additional dataset at level $l-1$. With a large number of items and a low support threshold the tree may become very wide, resulting in substantial memory consumption. The depth-first search approach addresses this problem by generating nodes path-wise, i.e., given a node p, a set of frequent nodes F_p is found and for each node $q \in F_p$ the procedure is recursively repeated until the last level of the tree is reached.

The pseudocode of the depth first algorithm is shown in Figures 4 and 3.13.

```
input  : root node r
         set of labels C
         set of items I
         support count threshold ξ̂
output: set of frequent nodes F
1  C_r ≔ C                                                    ◄ 1st level
2  CountSupport(C_r)
3  F_r ≔ Prune(C_r, ξ̂)
4  for each f ∈ F_r do
5  |  DepthFirst(f)
```

Figure 3.12: Pseudocode of the depth-first algorithm for multi-label, recurrent-items, associative-classification rule generation

```
input  : node p
         support count threshold ξ̂
output: set of frequent nodes F_p
1  if Parent(p) ∈ ∅ then
2  |  C_p ≔ I                                                 ◄ 2nd level
3  else if Parent(p) ∉ ∅ ∧ Parent(Parent(p)) ∈ ∅ then
4  |  C_p ≔ GenerateCandidates(S_p ∪ S_Parent(p))             ◄ 3rd level
5  else
6  |  C_p ≔ GenerateCandidates(S_p)                           ◄ levels 4 and up
7  for each T ∈ D_p do
8  |  IncreaseSupportAndCreateProjDatasets(T, C_p)
9  F_p ≔ Prune(C_p, ξ̂)
10 for each f ∈ F_p do
11 |  DepthFirst(f)
12 Delete(D_p)
```

Figure 3.13: Pseudocode of the recursive function `DepthFirst`

The depth-first algorithm begins with calling the `DepthFirst` function for each (frequent)

label. `DepthFirst` generates candidates following, again, the level-dependent procedure described in Section 3.4.2 (lines 1–6 in Figure 3.13). Since with the depth-first search it is impossible to determine a base node `Base(`p`)` for any p during the process of generating candidates (due to the fact that `Base(`p`)` is yet to be generated), the `GenerateCandidatesBF` function (see Figure 3.10) cannot be used. The original `GenerateCandidates` is used instead. `DepthFirst` is performed recursively for each frequent candidate (lines 10–11).

The maximum number of projected datasets held in the main memory at the same time depends on the number of levels and is equal to $\sum_{i=1}^{n} |C_{p_i}|$ where n is the number of levels in the currently mined path. Note that $|C_{p_i}| \leq |C_{p_{i-1}}|$.

The number of levels depends on the dataset and support threshold. The lower the threshold or the longer the patterns in the data, the bigger the number of levels and the longer the rules. Nevertheless, the maximum number of levels can never exceed the number of items, which creates a significant reduction in the number of projected datasets when compared to the exponential growth of projected datasets in the breadth-first search. A closer look at the complexity of these two algorithms is provided in the next section.

3.4.7 Complexity and limitations

The runtime and space complexities are discussed by comparing the breadth-first and depth-first algorithms as well as the *tree-based* Apriori algorithm, which was implemented for evaluation purposes (briefly described in Section 3.6).

Runtime complexity

Runtime complexity depends on the number of nodes (candidates) in the tree and the computation time needed for each node. For the sake of simplicity (without loss of generality), labels are treated in the same fashion as items.

The size of the tree depends on the number of items (and labels) and the *pruning factor*, i.e., the difference between the number of candidates and the number of frequent items. To further simplify the analysis, this difference is assumed to be constant for all groups of candidates, i.e., $C_p - F_p = k$ for each node p, where $k = const$.

This results in the following formula for the size of a tree $\mathcal{T}$ (excluding the root node):

$$|\mathcal{T}_k(n)| = \mathcal{F}_{k+1}(n+1) - 1, \tag{3.3}$$

where n is the number of items (and labels) and $\mathcal{F}_m$ is a generalized Fibonacci number,

which, in its combinatorial representation [83], is:

$$\mathcal{F}_m(n) = \sum_{i=0}^{\lfloor \frac{n+m-2}{m} \rfloor} \binom{n+m-2-(m-1)\,i}{i} \tag{3.4}$$

In the worst case scenario (e.g., a dataset with only one transaction or a set of exactly the same transactions) the size of the tree is $2^n - 1$ (which can be derived from (3.3) for $k = 0$). This exponential growth becomes weaker for greater values of k, and the complexity can be expressed as $O(c^n)$ for some constant c, where $c = 2$ for $k = 0$ and converges to 1 with increasing values of k.

If, however, pruning factor k is relative to the number of items n (for instance, $k = n/2$, i.e., a situation where half of first-level candidates are always frequent) it is expected that for certain relations of k and n the computational complexity will become lower than $O(c^n)$.

The recursive formula for the size of the tree is as follows:

$$|\mathcal{T}_k(n)| = \begin{cases} n & \text{if } n \leq k+1, \\ \sum_{i=1}^{n-k-1} |\mathcal{T}_k(i)| + n & \text{if } n > k+1, \end{cases} \tag{3.5}$$

This formula is derived directly from the visual representation of the ruleitem tree. Eliminating the recursion in (3.5) results in the following formula, which depends on the relation between k and n:

$$|\mathcal{T}_k(n)| = \begin{cases} n & \text{if } n \leq \frac{3k+1}{2}, \\ \frac{2n-3k-1}{2}k + n & \text{if } \frac{3k+1}{2} < n \leq 2k+1, \\ \sum_{i=1}^{n-2k-1} \mathcal{F}_{k+1}(n-2k-i)\,i + \frac{2n-3k-1}{2}k + n & \text{if } n > 2k+1, \end{cases} \tag{3.6}$$

It is observed from (3.6) that the bigger the difference between n and k, the bigger the tree growth rate. With constant k and increasing n the size of the tree grows from $O(n)$ (the first case) to $O(n^2)$ (the middle case) to $O(c^n)$ (the latter case, where the sum of generalized Fibonacci numbers mainly dictates the growth). Thus, if k is relative to n, the tree growth rate can decrease significantly. For instance, to continue with the previous example, for $k = n/2$ the tree growth becomes quadratic.

Analysis of the complexity of operations that must be performed at each node of the tree is reduced to the following operations: (1) generation, (2) support count, and (3) pruning. Both generation and pruning of a single node are constant. The complexity of counting support was already discussed in Section 3.4.4 and is proportional to $\frac{|D_p|}{|C_{\texttt{Parent}(p)}|}\,|T|$ for a node p and a transaction T, i.e., it takes $|C_{\texttt{Parent}(p)}|$ times less time than support counting in other algorithms [15].

The difference between the depth-first and breadth-first algorithms lies in the latter generating fewer candidates (though it yields additional comparisons). However, the asymptotic complexity of those two algorithms remains the same. As such, the majority of time is spent on the candidate test (counting support). The denominator $|C_{\texttt{Parent}(p)}|$ ranges from 1 to n with lower values on deeper levels of the tree. At the same time, the numerator $|D_p|$, bound by $\hat{\xi}$ and $|D|$, also decreases. Although the asymptotic upper bound of the candidate test is still $O(|D|)$, there is a substantial reduction in the number of candidate tests when compared to other algorithms that use tree-dataset projection. This is due to the decreasing size of the projected datasets D_p and performing aggregated (versus one node at a time) tests [116].

The total computational complexity of the proposed algorithms is $O(nm)$, $O(n^2m)$, and $O(c^nm)$ depending on the relation between k and n [116]. This varies from being linear to quadratic to exponential (the first two apply only if the pruning factor depends on the number of items) with respect to the number of items (and labels) with a constant number of transactions; and linear with respect to the number of transactions with a constant number of items (and labels). The tree-size-dependent complexity imposes certain limitations on MLACRI, which does not scale well in situations with long patterns in data (very similar transactions) and/or very small support thresholds. In both such situations the pruning factor will be close to zero which, in turn, will ”trigger” exponential growth in the number of operations. However, in text categorization the pruning factor is relatively high, and thus, the complexity is expected to be, at most, quadratic.

Space complexity

Due to the fact that MLACRI keeps projected datasets, its memory consumption is naturally higher than, for instance, the one of Apriori, in which memory consumption is proportional to $|D| + |\mathcal{T}|$. However, breadth-first and depth-first algorithms differ from each other in the number of projected datasets they maintain, and their respective memory consumption is proportional to:

- $|D| + |\mathcal{T}| + \sum_{p \in V_{max}} |D_p|$ for the breadth-first algorithm, and
- $|D| + |\mathcal{T}| + \sum_{i=0}^{\lfloor \frac{n+k}{k+1} \rfloor - 1} \sum_{q \in C_{p_i}} |D_q|$ for the depth-first algorithm,

where n, as before, is the number of items (and labels), p_i is an i-th node of the currently searched path in the tree (with p_0 being the root node), and V_{max} is a set of nodes at the widest level of the tree with the following cardinality (assuming the concept of pruning factor is as described above):

$$|V_{max}| = max_{0 \leq i \leq \lfloor \frac{n+k}{k+1} \rfloor} \left(\begin{array}{c} n+k-ki \\ i \end{array} \right). \tag{3.7}$$

Equation (3.7) is obtained from (3.4) and is equal to the maximum component of the sum for $F_{k+1}(n+1)$ in (3.4), which can be estimated as $O(c^n)$ for constant c, $1 < c < 2$. The number of nodes in C_{p_i} for each p_i in the path is $O(n^2)$. Based on these estimates and those made for the runtime complexity, the asymptotic complexity, in terms of memory consumption, is

- $O(m) + O(c^n) + O(c^n)O(m)$ and
- $O(m) + O(c^n) + O(n^2)O(m)$

for the breadth-first and depth-first algorithms, respectively [116]. Thus, the projected datasets increase memory usage (when compared to Apriori) exponentially and quadratically with an increasing number of items (and labels) for the breadth-first and depth-first algorithms, respectively.

3.5 Classification

Although association rules in models produced by MLACRI already consist of sets of classes to which a matching document should be assigned, usually more than one rule matches a given document. In order to overcome the problem of deciding which rule and/or how many of them should be taken into consideration when categorizing a document, the following thresholding strategies are proposed (see Section 2.1.1 for introduction to thresholding strategies):

RCut.% As opposed to RCut, which assigns top t classes to a document, RCut.% assigns a *fraction* of the total number of rules. Due to the variable number of rules matching a document, this strategy may assign it a different number of classes, which is desirable in classification with a variable number of classes.

SCut.global SCut.global is introduced to reduce the multi-parameter SCut strategy, which is impractical in classification with large number of classes, to a single parameter, which constitutes a global threshold for all the classes. Since in MLACRI all the classes on the right-hand-side of an association rule have the same score (the confidence of the rule), SCut.global in this case is rule-oriented.

There are two scoring variables that can be taken into consideration when applying the different thresholding strategies with MLACRI:

- the confidence of rules that match a document and

- the cosine measure that reflects the similarity between the rule and the document given their vector-space representations.

Whereas the former is document-independent, the latter shares similarities with scoring variables in other classifiers, such as SVM, in that it is calculated for each pair of class–document, or in the case of MLACRI, for each pair of rule–document.

Cosine measure is a value equal to an angle between two vectors. Given a document $\{w_1 i_1, \ldots, w_n i_n\}$ and a rule $\{o_1 i_1, \ldots, o_n i_n\} \rightarrow C$, where o_i and w_i are the numbers of occurrence of word i_i, the cosine measure is equal to $\arccos \angle(\vec{o}, \vec{w})$, where $\vec{o} = \{o_1, o_2, ..., o_n\}$ and $\vec{w} = \{w_1, w_2, ..., w_n\}$.

The entire process of both learning (building the model) and testing (verifying the effectiveness of the model) is illustrated in Figure 3.14.

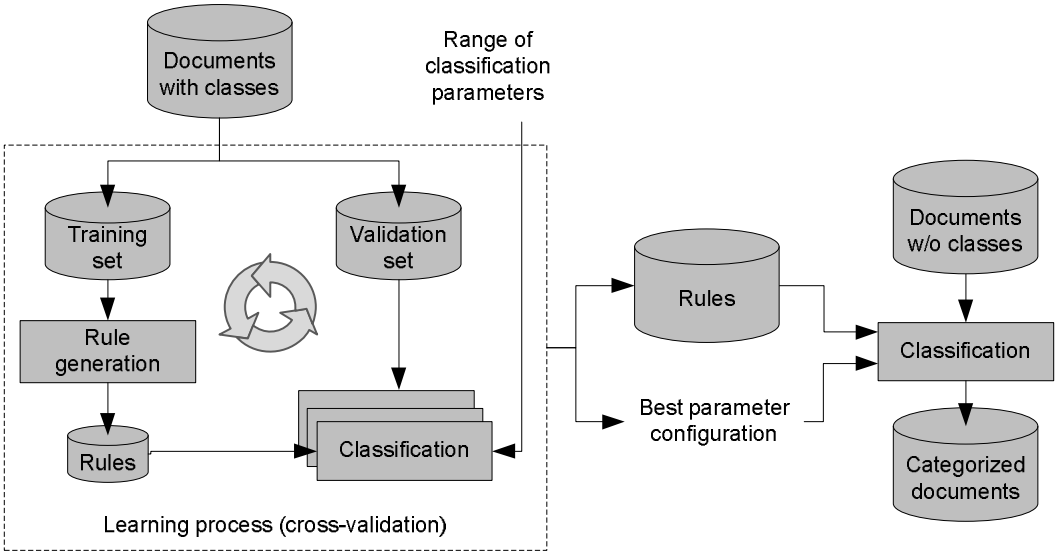


Figure 3.14: Classification learning and testing process

The documents that are used for building the model are split into two subsets: a *training* set and a *validation* set. The training set is used to generate a model, whereas the validation set is used to tune the thresholding parameters. The tuning is performed by testing a range of thresholding values in ten-fold cross validation (see Section 2.2.2) with the purpose of optimizing some classification efficiency measure (usually macro and micro F_1). The best value (the fraction of rules in the case of RCut.%, or the confidence or cosine measure in the case of SCut.global) tuned on the training–validation sets is expected to perform equally well with new, unseen documents.

The efficiency of classifying MEDLINE documents to MeSH is described in Section 3.6.4.

3.6 Evaluation

The evaluation of MLACRI is divided into quantitative and qualitative analysis. Section 3.6.1 describes the experimental setup used for both types of analysis. The empirical study of runtime and memory complexity is provided in Section 3.6.2. Section 3.6.3 discusses different characteristics of associative-classification rules with examples. The effectiveness of MLACRI compared to other relevant approaches is presented in Section 3.6.4.

3.6.1 Experimental setup

The experiments were performed on the OHSUMED [67] and RCV1 [90] corpora, commonly used in testing text classifiers (see the description of the datasets in Section 2.2.1).

The MeSH headings assigned to OHSUMED records have been modified such that the original 11-level structure was generalized to the second level resulting in the total of 114 class labels. From the three dimensions of labels available for the RCV1 collection, the 103-label "topics" were chosen as the most commonly used and adequate for the task. The two setups are referred to as *ohsumed-gen2* and *rcv1-topics*, respectively. Both datasets are summarized in Table 3.2.

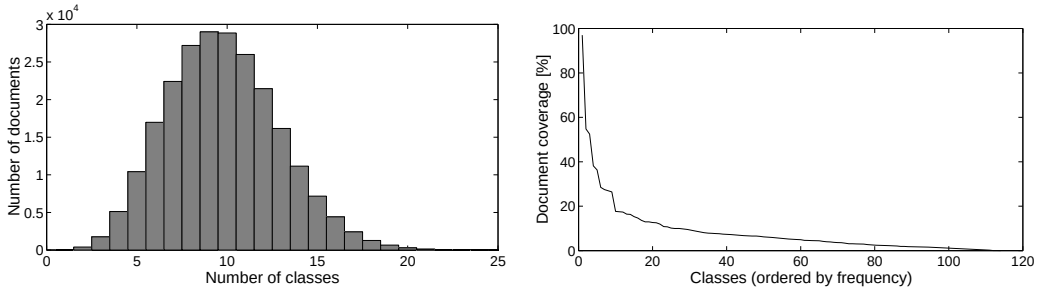
Table 3.2: Dataset statistics

		Number of labels			Number of words		
Dataset	Size	Total	μ	σ	Total	μ	σ
ohsumed-gen2	233,445	114	9.8	3.1	99,775	95.6	40.62
rcv1-topics	804,414	103	3.2	1.4	288,062	123.9	110.3

μ – average per transaction, σ – standard deviation

Both *ohsumed-gen2* and *rcv1-topics* are used in testing runtime and memory complexities. The *ohsumed-gen2* is also used in testing the effectiveness of MLACRI. The 233,445 records of OHSUMED were divided into two subsets: (1) 183,229 documents covering the years 1987–1990, which were used used as the training set, and (2) 50,216 documents from 1991, which were used as the testing set. This split conforms to other investigation with OHSUMED [92, 150, 127]. However, unlike the other attempts that used the testing set to tune the parameters of a classifier, the training set was further divided to perform ten-fold cross validation. This approach is more strict and produces non-overfitted models. A similar approach was used in [87] except that the authors performed only two-fold cross validation. The class distribution and frequency of the *ohsumed-gen2* dataset is shown in Figure 3.15.

The document preprocessing steps involved *stop word pruning* and *word stemming*. Stop



(a) (b)

Figure 3.15: (a) Distribution and (b) frequency of classes in the *ohsumed-gen2* dataset

words, which are words that appear frequently but are irrelevant with respect to classification, e.g., *a*, *the*, *of*, *at*, etc., were pruned from the documents to reduce the search space. Word stemming aimed to unify words by transforming lexical variants of words to single distinct entities. This operation was performed by employing the widely used Porter's algorithm [107].

3.6.2 Runtime and memory consumption

The time and memory complexity of the algorithm was empirically analyzed taking into consideration the different values of support threshold (which influences the number of items and labels) and dataset size as shown in Table 3.3.

Table 3.3: Experimental setup

Dataset	Size [k]	Support threshold [%]
ohsumed-gen2	12.5, 25, 50, 100, 200	2, 3, 4, 8, 12, 16, 20
rcv1-topics	25, 50, 100, 200, 400, 800	2, 3, 4, 8, 12, 16, 20

Apart from investigating the difference between the depth-first and breadth-first strategies, an additional algorithm, an Apriori-like version of an associative classifier, is also included in the comparison. This type of algorithm was chosen since an alternative pattern-growth-based method has been shown to generate only a subset of all frequent patterns [104] (see Section 3.2).

The Apriori algorithm works in a similar fashion to the one described in [95]. The main difference is that it is based on the ruleitem tree presented in this chapter, but does not incorporates the projection of datasets. This algorithm is referred to as *Apriori*, whereas the two versions of MLACRI, breadth-first and depth-first, are referred to as *MLACRI-BF* and *MLACRI-DF*, respectively.

Runtime was measured excluding the time needed for loading a dataset from a hard drive.

The loading time depends on the size of a dataset only, i.e., it is indifferent to parameters and algorithms chosen, and ranges from several seconds for the smallest datasets to several minutes for the biggest datasets used in the experiments.

The runtime and memory complexity with a fixed-size dataset of 100,000 transactions and various support threshold values is shown in Figures 3.16 and 3.17, whereas the algorithms' performance with various dataset sizes and a fixed 4%-support threshold is shown in Figure 3.18.

Support threshold

Figures 3.16 (a), (c), (e), and (g) show the runtime in the function of support threshold (notice the reverse order). Both MLACRI-BF and MLACRI-DF algorithms, whose plots almost overlap, outperform Apriori, which is especially visible in the lower values of the support threshold. Figures 3.16 (b), (d), (f), and (h) show the same runtime in function of the number of generated rules. Again, both MLACRI-BF and MLACRI-DF require significantly less time to generate the same number of rules than the Apriori algorithm. Although the three considered algorithms are characterized by the same linear asymptotic complexity with respect to runtime, linear extrapolation to 50,000 rules shows that it takes almost nine hours to generate rules using Apriori and only about 16 minutes using MLACRI.

The difference in runtime between Apriori and the two proposed algorithms is a result of storing projected datasets in MLACRI. The difference between MLACRI-BF and MLACRI-DF is the result of the lower number of candidates in MLACRI-BF. MLACRI-BF limits this number during the generation of nodes using information from the upper levels of the tree, which is not available in MLACRI-DF.

Memory characteristics, shown in Figure 3.17, are similar to those for runtime, i.e., memory consumption rapidly grows with the lower values of support threshold (Figures 3.17 (a), (c), (e), and (g)). However, in contrast to the runtime results, the faster algorithms, MLACRI-BF and MLACRI-DF, require more memory than the slowest, Apriori. The relatively large memory consumption in the case of both MLACRI-BF and MLACRI-DF is due to the storage of projected datasets, which is not the case with Apriori whose memory consumption depends solely on the number of generated rules.

Although MLACRI-BF and MLACRI-DF require comparable amounts of memory for higher values of the support threshold, the former requires significantly more memory for medium and low values of support. For MLACRI-DF, the number of projected datasets depends on the depth of the tree and not, as in the case of MLACRI-BF, its width. Since lowering support threshold results in creating wider, rather than deeper, ruleitem trees, MLACRI-DF is consequently characterized by higher memory consumption than MLACRI-BF.

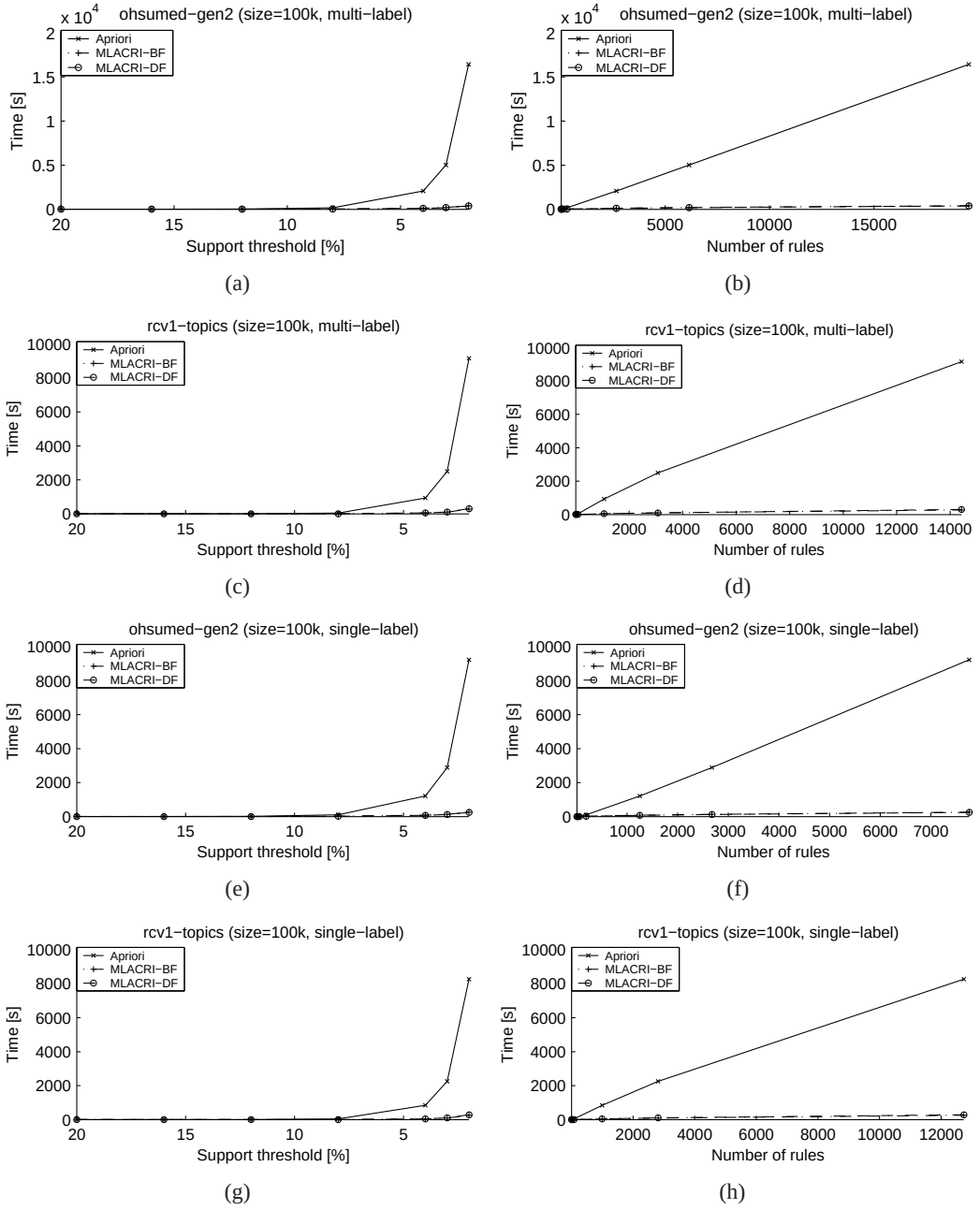


Figure 3.16: Comparison of the runtime of the algorithms MLACRI-BF, MLACRI-DF, and Apriori in the function of support threshold and the number of rules. (The plots for MLACRI-BF and MLACRI-DF overlap.)

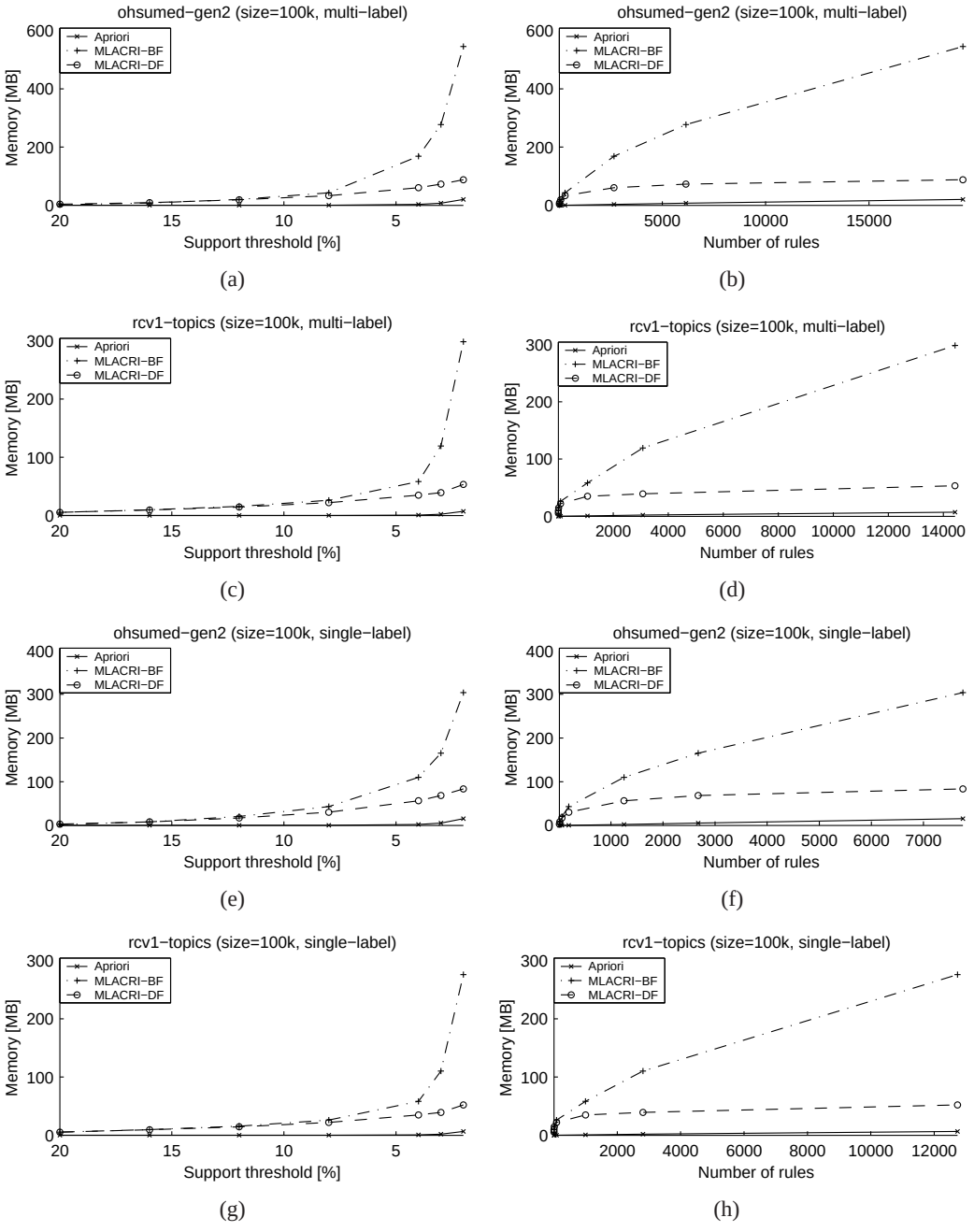


Figure 3.17: Comparison of the memory consumption of the algorithms MLACRI-BF, MLACRI-DF, and Apriori in the function of support threshold and the number of rules

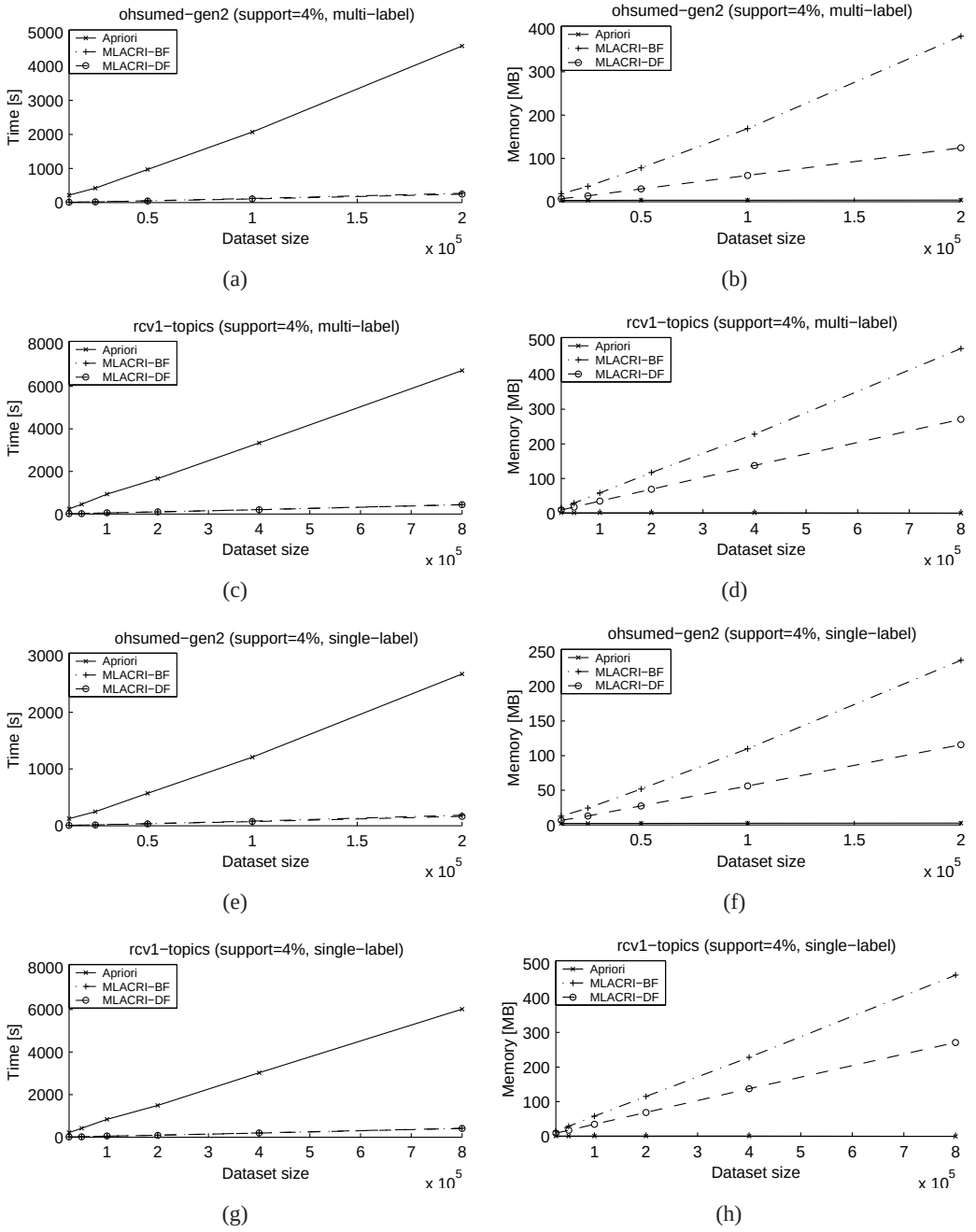


Figure 3.18: Comparison of the runtime and memory consumption of the algorithms MLACRI-BF, MLACRI-DF, and Apriori in the function of dataset size. (The plots for MLACRI-BF and MLACRI-DF overlap in the left-hand-side figures.)

The computational limitations of MLACRI discussed in Section 3.4.7 are confirmed by the experiments. Either a low support threshold or long patterns in data, i.e., when very similar transactions yield a large number of patterns even if a support threshold is high, result in numerous nodes in the tree and potentially large projected datasets (or references to the generic dataset as discussed in Section 3.4.5). Nevertheless, setting a reasonable support threshold is left to user's discretion (e.g., building a rule-based classification model that consists of more rules than transactions may raise some doubts about the usefulness of such a model).

Dataset size

Figure 3.18 shows the runtime and memory complexity with different sizes of the datasets. The results indicate a linear relation to the number of transactions (as estimated in Section 3.4.7). Similarly to the previous results, MLACRI-BF and MLACRI-DF show comparable scalability and are superior to Apriori with respect to runtime (Figure 3.18 (a), (c), (e), and (g)). For instance, it takes about 237 seconds for Apriori and only about 12 seconds for MLACRI-DF to generate rules from 25,000 transactions on *rcv1-topics* and a support threshold of 4%. For larger 800,000-transaction dataset Apriori needs almost two hours compared to about 7 minutes in the case of MLACRI.

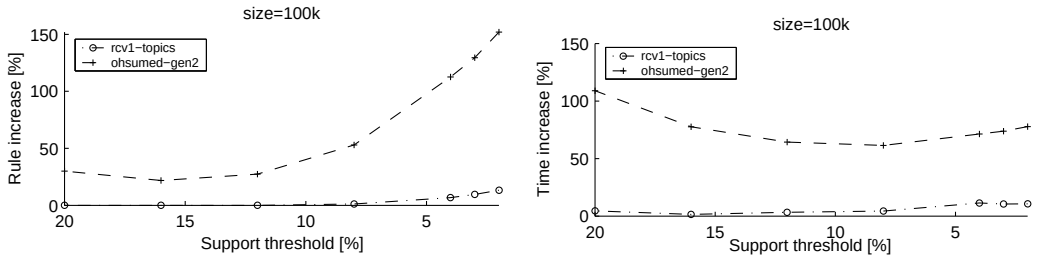
On the other hand, memory consumption diagrams (Figure 3.18 (b), (d), (f), and (h)) show an opposite relation. Memory usage for Apriori is almost constant, whereas for MLACRI the required memory linearly increases with the increasing number of transactions.

Assuming uniform distribution of words in a dataset, the number of rules generated with the same support threshold should be virtually the same for any subset of the dataset. This implies that the ruleitem tree structure be almost identical. That is why Apriori uses nearly the same amount of memory for different dataset sizes. Since MLACRI requires storing projected datasets, its memory consumption increases linearly with the size of the generic dataset.

Single- vs. multi-label rule generation

The results discussed in the previous sections show that the three considered algorithms applied to both single-label and multi-label rule generation have the same asymptotic complexity. Figure 3.19 shows a summarized comparison of the cost of generating multi-label rules in relation to single-label rules with respect to the number of rules generated and runtime.

With the same support threshold the two types of rule generation methods differ in the



(a) (b)

Figure 3.19: Increase (a) in the number of rules between multi-label and single-label rules, and (b) in runtime to generate multi-label rules in relation to single-label rules

number of generated rules. This difference is especially visible with the low values of support threshold. For instance, setting the support threshold value to 4% on the ohsumed-gen2 dataset results in generation of about 110% more multi-label rules than single-label rules with only about 80% increase in runtime. (The difference between the two datasets used in the experiments is a consequence of their class label distribution, i.e., the average number of labels per transaction is significantly higher in the ohsumed-gen2 dataset than in the rcv1-topics dataset (see Table 3.2).) This shows that the generation of the multi-label rules, that are supersets of single-label rules, can be accomplished with relatively low cost when compared to the generation of the single-label rules.

3.6.3 Analysis of generated rules

Table 3.4 shows several rules for each of the dataset used in the experiments.

The presented rules were chosen to show the variety of forms they can take and to be comprehensible to non-experts. All item names are shown in their preprocessed, stemmed form, e.g., *studi*, *compan*, or *pric* (which in some cases may be ambiguous). The table includes rules with one item and one label (rules (iii), (v), and (ix)), with multiple items and one label (rules (i), (ii), (viii), and (xi)), with one item and multiple labels (rules (vi) and (vii)), and finally multiple items and labels (rules (iv), (x), and (xii)). As expected, introducing recurrent items increases the confidence of the rule, decreasing support at the same time (rules (iv) and (vi)). Rules with high support and low confidence (such as rule (vii)) indicate that the left-hand-side words are commonly used with the vast number of classes and therefore might be candidates to be included on the list of *stop words*, i.e., the list of words that appear frequently but are irrelevant to classification. Clearly, the most valuable rules, from a *predictive* point of view, are those with high support and high confidence. However, in practice such rules are of doubtful usefulness from a *descriptive* point of view, as they usually carry obvious knowledge (see rule (ix)). Thus, experts may be more interested in rules with high confidence disregarding their support at the same time.

Table 3.4: Examples of associative-classification rules in (a) *ohsumed-gen2* and (b) *rcv1-topics*

	Rule*	Support	Confidence
(a) ohsumed-gen2			
(i)	rat(2) → Animals	5.30%	99.96%
(ii)	effect, studi, control → Animals	4.29%	99.35%
(iii)	children → Persons	4.64%	95.71%
(iv)	tumor(2) → Animals, Neoplasms	4.11%	93.80%
(v)	tumor → Neoplasms	6.13%	89.09%
(vi)	tumor → Animals, Neoplasms	6.10%	82.94%
(vii)	studi → Persons, Animals, Investigative Techniques	12.26%	30.75%
(b) rcv1-topics			
(viii)	million, net, profit → Corporate/Industrial	4.02%	92.28%
(ix)	compan → Corporate/Industrial	21.84%	78.50%
(x)	net, profit → Corporate/Industrial, Performance	4.19%	78.09%
(xi)	net, profit → Performance	4.19%	78.09%
(xii)	market, pric → Commodity markets, Markets	4.00%	31.03%

*) number in parentheses denotes recurrence

3.6.4 Classification

The effectiveness of classification was tested by performing a number of experiments that included different thresholding strategies and scoring methods.

The results of evaluating the different thresholding strategies on the *ohsumed-gen2* dataset are presented in Table 3.5.

MLACRI was compared against SVM [71], a binary classifier shown to exhibit superior effectiveness to other methods [90], which was adapted to multi-label classification by combining results from n independent classifications, where n is the number of classes. In this scenario each binary classification aims at categorizing documents to either the positive or negative class in the *one-vs-all* fashion, i.e., one class is treated as the positive while the remaining classes are treated as the negative[3]. The SVM model learning process was used in a setup similar to the one presented in [90].

The thresholds for both MLACRI and SVM were allocated automatically in the process of ten-fold cross validation (see Section 3.5) and tuned for best macro F_1 and best micro F_1 separately.

[3]Another adaptation of binary classifiers to multi-label classification is the *one-vs-one* strategy, where separate classifiers are built for each pair of classes. However, due to the large number of such combinations, this strategy is not suitable for the considered task.

Table 3.5: Comparison of multi-label classification performance on *ohsumed-gen2* between two baseline classifiers, MLACRI, its version with suppressed recurrent item information MLAC~~RI~~, and SVM. Precision (P), recall (R), and F_1 are reported in percentages.

			macro-averaging			micro-averaging		
Method	Thresholding	Scoring	P	R	F_1	P	R	F_1
Baseline			9.0	8.5	8.7	28.0	26.4	27.2
MLAC~~RI~~	RCut.%	confidence	40.8	55.3	42.0	53.5	58.8	56.0
MLAC~~RI~~	SCut.global	confidence	41.9	52.9	41.7	53.4	58.3	55.8
MLACRI	RCut.%	confidence	44.9	56.6	45.9	55.4	59.8	57.5
MLACRI	RCut.%	cosine	43.8	59.0	45.5	57.8	57.7	57.7
MLACRI	SCut.global	confidence	43.1	58.1	45.4	54.8	60.9	57.7
SVM	RCut		8.9	15.6	11.1	27.3	32.0	29.5
SVM	SCut.global		70.4	63.9	65.9	78.9	69.6	74.0

In order to investigate the influence of recurrence of items in transactions on classification performance, the comparison also includes a version of MLACRI, referred to as MLAC~~RI~~, where this feature is suppressed. Additionally, a baseline showing the random assignment of classes according to their distribution in the training set is drawn[4]. When compared to the baseline, MLACRI increases micro-averaged F_1 over two times, and macro-averaged F_1almost five times.

Considerable differences between the micro- and macro-averaged measures is the result of highly unbalanced classes (see Figure 3.15(b)). For instance, the most frequent class appears in 97.02% of the *ohsumed-gen2* documents, whereas the least frequent, in only 0.02%. The higher values of the micro-averaged measures over the macro-averaged measures suggest that the classifiers model frequent classes more effectively.

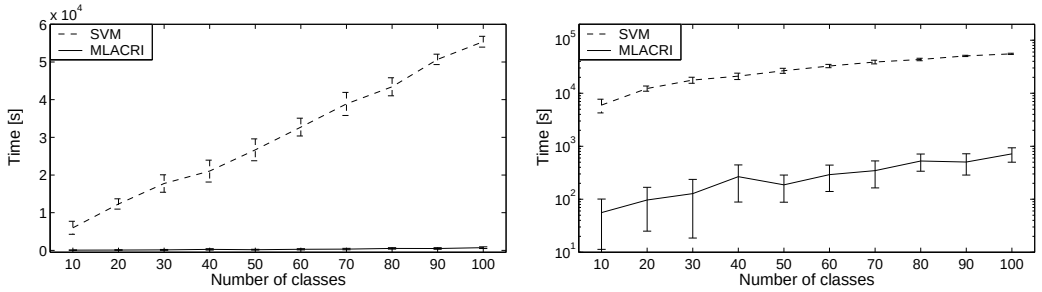
The results show that the introduction of recurrent items improves the performance of MLACRI. The difference is especially notable for macro-averaged F_1, which suggests that MLACRI performed comparably well for all the classes (regardless of their distribution in the documents).

Insofar as the thresholding strategies did not have a big impact on MLACRI, SVM was significantly affected. Low results for SVM with the RCut strategy shows that this strategy is inferior to RCut.%. RCut assigns a fixed number of classes to a document, and this strategy proved to be of no use in classification with a variable number of classes per document. Moreover, since SVM does not produce a variable number of classes, as opposed to MLACRI, RCut.% cannot be used with this classifier.

[4]The random assignment of classes involves randomly selecting a number of documents for each class from the testing set proportionally to the number of documents assigned to a particular class in the training set. This procedure was repeated ten times and the results averaged.

Nevertheless, SVM with SCut.global strategy proved to be a superior method of classification and outperforms all other methods by a significant margin. It was shown, however, that this margin can be reduced by using a two-stage learning strategy. Antonie [17] used association rule mining to build a first-stage classification model and applied another learning strategies, such as NN and kNN, to automatically learn the scoring scheme used in the final model of classification. The method was evaluated on a set of 12,202 documents of the Reuters-21578 collection [8] boasting overall performance comparable to SVM.

It has also been argued [144] that binary classifiers, such as SVM, are not scalable to problems such as the classification of MEDLINE documents to the MeSH terms. This stems from the fact that, in order to perform in the multi-label environment, binary classifiers have to be trained for each class separately. Given the high complexity of the learning process, SVM is limited to problems with a relatively low number of classes, which is not the case with the large MeSH thesaurus. In order to show the difference in class scalability, an analysis of the time needed to develop models with various number of classes has been performed, the results of which are shown in Figure 3.20. Due to the differences in class distributions, the time characteristics presented in the figure were obtained by randomly choosing the corresponding number of classes from the pool of all the available classes. As before, the procedure was repeated ten times and the results averaged.



(a) (b)

Figure 3.20: Runtime of building models with different number of classes for SVM and MLACRI on the *ohsumed-gen2* training set (183,229 examples) in (a) linear and (b) logarithmic scales. Vertical bars denote standard deviation.

On average, the time needed to produce the models with SVM is two orders of magnitude bigger than with MLACRI. For instance, SVM takes 15 h 23 min. to build models for 100 classes, while MLACRI tackles this task in less than 4 min. Linear extrapolation suggests that if the MeSH terms used in the classification task were generalized to the third level, the total of 1,734 third-level classes would take SVM over 263 hours compared to MLACRI's three hours. Moreover, the non-monotonic characteristic of MLACRI demonstrates that the method is rather insensitive to the number of classes and depends mostly on their distribution in documents.

Recently, Trieschnigg *et al.* [144] showed a comparative evaluation of MetaMap [18], MTI [19], a pattern-based approach [126], and kNN by attempting to categorize 1,000 MEDLINE documents to MeSH and analyzing different distributions of MeSH terms in the documents. The best results were reported with 196 MeSH terms that appeared in at most 5,000 documents, which is the closest test that can be compared to the evaluation setup presented in this section. The winner, kNN, achieved micro F_1 reported at a level of 56%, which is an almost identical result to MLACRI. However, it is worth pointing out that the evaluation strategy adopted in [144] overfits the (already incomparably small) testing set, i.e., the authors did not perform a proper cross-validation. Moreover, the small class sizes (at most 5,000 documents) suggest that the documents consisted of very selective sets of terms, which fitted the testing set well. As long as the readability of models is considered, kNN, whose instance-based model consist of virtually complete documents, is inferior to the concise nature of the association rules produced by MLACRI.

A separate evaluation, not directly pertinent to the study presented in this chapter, was performed to compare MLACRI to other descriptive classifiers in a single label classification task. MLACRI was shown to outperform all reported classifiers including associative-classification-based MMAC [142] and CBA [95], decision-tree-based PART [56], and rule-based RIPPER [40]. The results are reported in [116].

3.7 Conclusions

The MLACRI classifier addresses the generation of "truly" multi-label association rules, i.e., a model consists of rules with multiple classes generated in a single run. The experimental results showed the superior performance of MLACRI over the Apriori-like algorithm with respect to the runtime. As expected, memory consumption is bigger for the proposed tree-projected algorithms than for the Apriori-like algorithm due to the use of projected datasets. Although this difference is considerable for the breadth-first search, the memory characteristics indicate significantly better (smaller) memory consumption for the depth-first algorithm. At the same time, there is only a marginal difference between the depth-first and the breadth-first algorithms in terms of runtime, with the latter being superior to the former. This nominates the depth-first algorithm as the recommended solution. The depth-first and breadth-first algorithms scale linearly with respect to the dataset size for both runtime and memory usage. Although complexity with respect to the number of items may become exponential (for frequent and long patterns), several tree-based techniques were presented to significantly slow down this growth.

Although the effectiveness of MLACRI was inferior to SVM, it was shown that SVM, which is a binary classifier and requires an adaptation to perform classification with multiple

classes, does not scale well to problems that involve large number of classes.

When compared to a baseline, MLACRI increased the effectiveness of classification by over two times with respect to micro F_1 and by almost five times with respect to macro F_1. Considering the recurrence of words in documents additionally improved the predictive quality of the classifier, especially with respect to macro F_1.

The performance was evaluated using a *second-level generalization* of the MeSH structure, and as such, MLACRI (or any other known classifier for that matter) cannot be perceived as a *fully* automatic method of classification given the task. Classification of the full set of 24,000 hierarchically distributed MeSH terms remains an open subject and fully automatic methods are yet to be developed.

Chapter 4

Entity Relationship Extraction

4.1 Introduction

Extracting entity relations from textual corpora is one of the information extraction tasks that has been gaining researchers' attention, motivated mainly by an overwhelming growth in the number of scientific articles and the need for tools that concisely summarize and present the knowledge embedded therein. The extracted relations between annotated entities are used in a variety of applications, such as querying-answering systems, often built on top of ontologies which allow storing well-defined and well-structured data (e.g., [37, 14]), or in biomedicine where proteins, genes, and cells are examples of entities, and the task of extraction is to find interactions between them [69, 46, 45, 100, 152, 35].

Extracting binary relations have been attempted using several different techniques, most notably shallow phrase structure parsers, typed dependency parsers, and machine learning (often combined with either shallow or deep parsing).

Typed dependency parsers assign a grammatical dependency/role between a pair of tokens. For instance, the Stanford dependency parser [49] identifies such pairs and assigns one from over 50 grammatical dependencies to each pair. A generated set of grammatical dependencies from a typed dependency parser is further used to build binary relations (involving one or more grammatical dependencies) between a pair of entities, usually by traversing *dependency graphs* or by creating a set of logic programming rules. The biggest disadvantage of using dependency parsers is that there is no standard set of grammatical dependencies [39]. They differ in terms of number, granularity, direction, and even the part of speech (POS) of the constituents on both sides of seemingly identical dependencies.

On the other hand, the Penn Treebank Project, a vast collection of manually annotated constituent trees, created a *de facto* annotation standard, which has been followed by major

leading constituent parsers created to date. Intuitively, given that a dependency parser produces dependency roles based on information from the phrase decomposition of a sentence (either explicitly or implicitly), relation extraction systems based on pure phrase decomposition (coming from a constituent parser) should perform no worse than systems based on dependency parsing, as long as the constituent-parsing-based systems are capable of capturing complete binary relations between entities embedded in a sentence, as opposed to only finding local relations between tokens in a sentence (as it is the case with dependency parsers).

Although the significance of constituent trees in extracting relations in sentences was recognized over a decade ago [77, 120], proper tools exhibiting the sufficient flexibility and expressiveness necessary to discover a correct and complete set of relations have yet to be designed. This investigation addresses this issue and proposes a method for extracting relations between two entities based on a manually tailored set of constituent pattern trees. As a result, the major contributions of this work include the development of the *constrained constituent tree inclusion algorithm with regular expression matching* as well as a syntax of pattern trees together with a set of patterns.

Depending on the number of words connecting two entities in a relation and their POS, the relations are grouped into several categories. They vary from relations including a single verb (e.g., *<inflammatory cytokine> initiate <autocrine regulatory mechanism>*[1]) to the more elaborate including verbs and prepositions (e.g., *<LTR mutation> allow to replicate during <acute phase of viral infection>*).

The remainder of this chapter is organized as follows: Section 4.2 presents related work in the field of relation extraction, which is followed by defining the problem of extracting relations from constituent trees in Section 4.3. A solution to the given problem is proposed in Section 4.4, where a new ordered-tree-inclusion-based algorithm together with a new set of patterns are presented. The proposed solution is evaluated experimentally as described is Section 4.5, whereas Section 4.6 summarizes the chapter.

A version of this chapter has been submitted for publication [118].

4.2 Related work

Extracting entity relations in sentences have been attempted using a variety of methods, which involve shallow parsers, dependency parsers, binary trees, and, to a certain extent, constituent trees.

Cimiano *et al.* [37] used a shallow parser (or chunker) to process sentences and a set of

[1] Annotated entities are denoted by enclosing them in triangle brackets ('<' and '>')

flat patterns to extract relations from the shallow-parse trees. They used GENIA [79] as an input corpus, the CASS chunker [11] as a shallow parser, and two simple patterns, a noun phrase followed by a verb, followed by a noun phrase optionally preceded by a preposition. They also attempted to find the appropriate generalization level for the relations obtained in the GENIA ontology [79].

Rinaldi *et al.* [121, 123, 122] used Pro3Gres [129], a typed dependency parser, the output of which they further converted into binary relations. The conversion involved combining several dependencies, generated during parsing from a sentence, in a set of Prolog rules. Querying the rules directly returned the relations. The authors evaluated the performance of the linguistic performance of the parser on a subset of GENIA and the authors' ATCR corpus, a 147-sentence set of annotated MEDLINE abstracts, considering relations that involved one of a small set of verbs. They further evaluated the biological significance of the obtained results, reporting precision and recall separately for the pairs subject–verb and verb–object.

A similar dependency-like approach was used by Ciaramita *et al.* [36]. The authors used the Charniak parser and built dependency graphs between the tree constituents. They further formed verb relations between the annotated entities in GENIA by selecting unidirectional paths in graphs limited by a handful of dependency constraints, that connect the subject entity through a verb to the object entity. Similarly to [37] they were interested in generalizing the relations to an appropriate GENIA ontology level, and thus considered only the categories of annotated entities.

Abulaish and Dey [14] proposed a novel tree-structure-like approach. They decomposed sentences and encoded them as binary trees. Each of the nonterminal nodes of a tree stored a verb, whereas the terminal nodes contained the remaining parts of the sentence directly preceding and following a parent verb. This allowed the authors to create relations, which were subjected to a frequency-based feasibility study on GENIA performed by replacing the annotated entities with the GENIA categories.

An output of a constituent parser was used by Jang *et al.* [69] to extract protein-protein interactions (PPI) from MEDLINE. The authors used the Stanford parser and a handful of heuristics to traverse constituent trees in search for relations between annotated proteins.

Daraselia *et al.* [45] showed a pattern-based approach to extracting PPIs. A set of manually tailored patterns was matched against sequences of sentence words. The patterns, encoded as regular expressions, consisted of a set of named, identified beforehand, interactions, i.e., the patterns matched only those sequences in sentences that involved certain predefined words. Due to the fact that PPI interactions are well known and identified, the process of extracting these interactions in text is seen as a simplified problem of extracting unknown relations, as presented in this chapter.

Banko *et al.* [21, 22] is an example of a hybrid approach that combines both machine learning and linguistic parsing. Their self-supervised training is based on a set of heuristics that was applied to Penn Treebank. The captured dependencies obtained via syntactic parsing and typed dependency labeling were used with the heuristics to identify positive and negative examples. The categorized examples were further used to train a Naiv̈e Bayes (NB) or Conditional Random Fields (CRF) classifier based on the examples' features such as part of speech (POS), the number of tokens, the number of stop words, etc. Once trained, the tool is claimed to efficiently extract a huge number of sentences (predominantly the method aims at extracting relations from the web), as at that point it only requires a shallow phrase structure of sentences (obtained from a *phrase chunker*) to proceed. In order to evaluate the method the authors manually identified several relation categories in a set of 500 sentences fetched from the web [22] and reported the performance of their approach comparing both the NB and CRF-based methods.

In Section 4.5 the proposed constituent-tree, pattern-based approach is evaluated by comparing it to the work of [37], [122], and [22], which represents shallow-parser-based, dependency-parser, prolog-rule-based, and dependency-parser, machine-learning-based efforts to extract binary relations between entities, respectively.

4.3 Problem definition

Given a constituent tree representing a sentence, the task of extracting entity relations can be loosely defined as finding *tree inclusions* (formally defined below), which encompass the words that constitute a relation and exclude those that do not. This task is accomplished by creating a set of pattern trees and running a tool which returns inclusions matching the patterns in the constituent trees.

An example of a constituent tree is given in Figure 4.1[2]. Given the objectives, the entity relation extraction process should result in three relations corresponding to three tree inclusions as follows:

1. *<Spi-B> binds <DNA sequences>* – nodes 1, 2, 3, 4, 5, 6, 7, 8, 9;
2. *<Spi-B> activates <transcription>* – nodes 1, 2, 3, 4, 17, 18, 19, 20;
3. *<DNA sequences> containing <5'-GGAA-3'>* – nodes 1, 4, 5, 7, 8, 9, 10, 11, 12, 15.

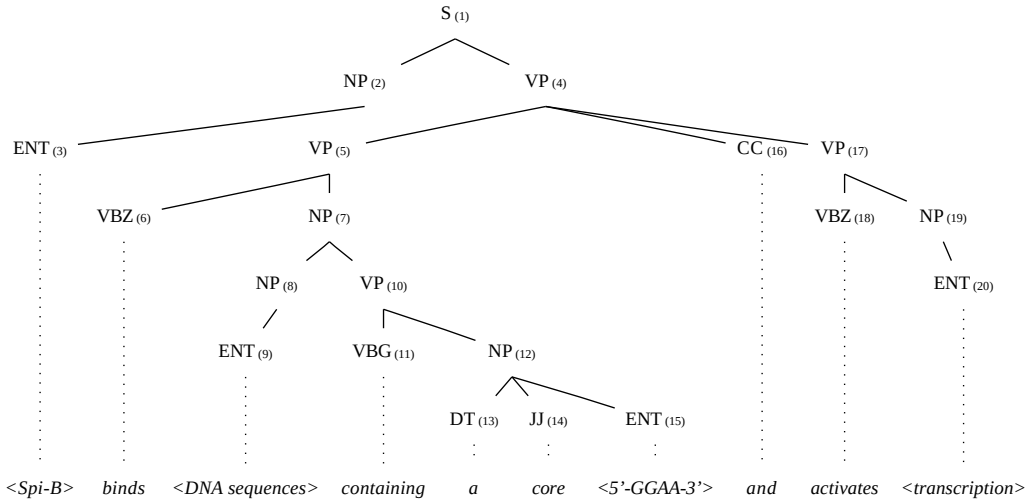
The first two relations have similar tree inclusions in terms of structure and labels, whereas the last relation differs from the former with regard to both structure (more nodes) and labels

---

[2] Although the Penn Treebank Project uses *round* brackets, throughout this chapter *square* brackets are used instead to avoid confusion with the regular expression syntax (explained later in the text)

```
[S
  [NP [ENT <Spi-B>]]
  [VP
    [VP [VBZ binds]
      [NP
        [NP [ENT <DNA sequences>]]
        [VP [VBG containing]
        [NP [DT a] [JJ core] [ENT <5’-GGAA-3’>]]]]]
    [CC and]
    [VP [VBZ activates]
      [NP [ENT <transcription>]]]]]
```

(a)



(b)

Figure 4.1: Example of a constituent tree in (a) the bracketed form and (b) its corresponding graph representation. The `ENT` label is introduced to denote a leaf node that contains an annotated entity and as such is not part of the Penn Treebank annotation tag set.

(e.g., VBG instead of VBZ).

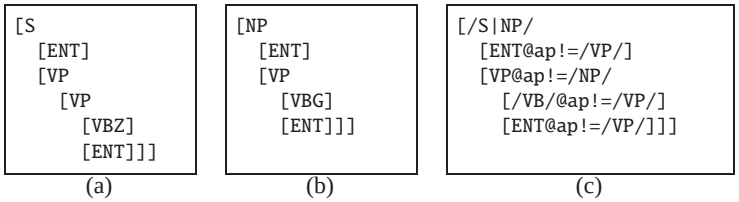
Extracting tree inclusions from trees has been already studied and several algorithms for solving the *ordered tree inclusion problem* have been proposed [77, 120]. The problem is defined as follows:

Definition 4.3.1 (Ordered tree inclusion) *Given a labeled ordered pattern tree P and a labeled ordered target tree T, the problem of* ordered tree inclusion *is to find a mapping function f from P to T such that $\forall v, v_1, v_2 \in P$:*

1. v_2 is a descendant of $v_1 \leftrightarrow f(v_2)$ is a descendant of $f(v_1)$,
2. $v_2 > v_1 \leftrightarrow f(v_2) > f(v_1)$,
3. v and $f(v)$ have the same labels.

In the above and following definitions the greater-than and lower-than operators ('>' and '<') refers to the order of nodes in a tree, where *order* is understood as in traversing the tree in *preorder* (an example is given in Figure 4.1(b), where the nodes' unique identifiers are allocated in preorder). When referring to the order between two nodes v_1, v_2, it is stated that v_1 *precedes* (*follows*) v_2 if $v_1 < v_2$ ($v_1 > v_2$). Subsequently, *previous* and *next* are used to express *immediately preceding* and *immediately following*, respectively.

Using the example given in Figure 4.1, the three relations can be extracted solving the ordered tree inclusion problem with the two patterns presented in Figures 4.2(a) and (b). The first pattern, however, will also produce an incorrect relation, *<Spi-B> binds <5'-GGAA-3'>*. Careful analysis of the tree reveals that, in fact, it is *not* possible to create a pattern or a set of patterns that when used to solve the problem specified in Definition 4.3.1 yields the correct and complete set of relations embedded in that particular sentence.



(a) (b) (c)

Figure 4.2: Examples of (a,b) ordered tree inclusion patterns and (c) a constrained constituent tree inclusion pattern

Although the tools solving the ordered tree inclusion problem are capable of extracting relations in complex sentences (e.g., sentences with multiple subordinate clauses) boasting high recall, they fail to show satisfactory levels of precision for practical use. Moreover, in practical applications usually only the part of speech of words constituting a relation is

specified, while other grammatical properties of words, such as tense in case of verbs or number in case of nouns, are neglected. This lowers the value of the ordered tree inclusion even more since more patterns are needed to account for all possibilities.

The above shows that inclusion patterns need to be more flexible so that a single node covers more than one type of target nodes. At the same time certain cases also require limiting the "greediness" of inclusion patterns by imposing some tighter constraints onto the ancestor–dependent relations between nodes.

4.4 Proposed solution

This section describes the proposed solution to extracting entity relations in sentences. Section 4.4.1 sets up the theoretical background to an algorithm described in Section 4.4.2; whereas Section 4.4.3 presents the syntax of patterns required by the algorithm. Both the algorithm and the patterns are the main components of the method presented in this work, and a complete processing pipeline that uses the two is given in Section 4.4.4.

4.4.1 Constrained constituent tree inclusion problem

In order to accommodate the aforementioned flexibility of patterns, an algorithm that solves the problem of *constrained constituent tree inclusion with regular expression matching* (defined below) is proposed.

The constrained constituent tree inclusion problem necessitates extending the properties of both a pattern node and a target node. Although from the tree structure perspective a word and its POS are treated as separate nodes, for the obvious one-to-one parent-child relation no such distinction is made and, instead, both are treated as one node where the POS is a *label* and the word is a *content*, e.g., in `[VBZ activates]`, `VBZ` is a label and `activates` is a content; whereas nonterminal nodes have a label only.

The requirements imposed on a pattern node call for a more elaborate set of properties as follows (all of the properties, except *label*, are optional):

label a string or regex[3] representing node's label or a class of labels (e.g., `VBZ`, `VBG`, `/^VB/`),

content (leaf nodes only) a string or regex representing a token (word) or a class of tokens (e.g., `not`, `cannot`, `/(can)?not/`),

[3]Following the software developers community, a sequence (or string) of characters is simply referred to as *string* and a regular expression as *regex*. Regexes are presented in the Perl-like notation (see Section 2.1.2).

ancestor path pattern constraint a string or regex representing a path of ancestors,

first (last) sibling path pattern constraint a string or regex representing a path of preceding (following) siblings,

child-of constraint a flag that, when set, forces the child-of relation,

sibling constraint a flag that, when set, forces the sibling relation.

Definition 4.4.1 (Constrained constituent tree inclusion) *Given an ordered pattern tree P with nodes having the aforementioned properties and a target constituent tree T, the problem of* constrained constituent tree inclusion with regular expression matching *is to find* all *mapping functions f from P to T such that $\forall v, v_1, v_2 \in P$:*

1. tree structure conditions:
   (a) order: $v_2 > v_1 \leftrightarrow f(v_2) > f(v_1)$,
   (b) ancestor–descendant: v_2 is a descendant of $v_1 \leftrightarrow f(v_2)$ is a descendant of $f(v_1)$,
   (c) child-of contraint: v_2 has a child-of constraint and v_2 is a child of $v_1 \rightarrow f(v_2)$ is a child of $f(v_1)$,
   (d) sibling constraint: v_2 has a sibling constraint and v_2 is the next sibling of $v_1 \rightarrow f(v_1)$ and $f(v_2)$ are siblings.
2. string conditions:
   (a) the label of v matches the label of $f(v)$,
   (b) v is a leaf and v has a *content* $\rightarrow$ the content of v matches the content of $f(v)$,
   (c) v_2 is a child of v_1 and v_2 has an *ancestor path pattern constraint* $\rightarrow$ the ancestor path pattern of v_2 matches the path between $f(v_2)$ and $f(v_1)$,
   (d) v has a *first (last) sibling path pattern constraint* $\rightarrow$ the first (last) sibling path pattern of v matches the path between $f(v)$ and the first (last) child of the parent of $f(v)$,
3. optional node: $\nexists f(v)$ and v is *optional* $\rightarrow$ v and its descendants are excluded from P for the map f.

To continue with the example given in Figure 4.1, as it was mentioned earlier, it is not possible to create a set of patterns that works with the *ordered tree inclusion problem* (as given in Definition 4.3.1) and yields the correct and complete set of relations. The two proposed patterns (Figures 4.2(a) and (b)) captured all the relevant relations and one *irrelevant* relation. On the other hand, solving the problem of *constrained constituent tree inclusion* (as

given in Definition 4.4.1) with the pattern shown in Figure 4.2(c) results in the complete and correct set of relations. The pattern combines the two previous patterns by incorporating regexes in the first node (`/S|NP/`) and the fourth node (`/VB/`), as well as imposes an ancestor path pattern constraints (`@ap!=/NP/` and `@ap!=/VP/`) on nodes 2–5. The constraints, which read "the *ancestor path* must not contain `NP`/`VP`", guarantee to capture all the relevant relations, while omitting the irrelevant relation *<Spi-B> binds <5'-GGAA-3'>*.

The full syntax of pattern trees is presented in Section 4.4.3.

4.4.2 A constrained constituent tree inclusion algorithm

In this section a *backtracking* algorithm is proposed. The algorithm takes a pattern tree P and a target tree T as input and gradually (in the order of nodes in P) builds a (possibly empty) set of maps from P to T satisfying Definition 4.4.1. The algorithm is inspired by the solution proposed for the ordered tree inclusion problem given in [120].

Table 4.1: Pattern tree node references

`Parent(`x`)`	the parent node of x
`PreviousPreorder(`x`)` and `NextPreorder(`x`)`	the previous and next node (in preorder) of x, respectively
`PreviousSibling(`x`)` and `NextSibling(`x`)`	the previous and next sibling of x, respectively
`FirstChild(`x`)`	the first (in preorder) descendant of x
`LastLeaf(`x`)`	the farthest (in preorder) descendant of x, or x itself if x is a leaf

The algorithm `FindMatch` is depicted in Figure 4.3, whereas Table 4.1 describes the necessary references related to a node. Furthermore, in order to ease the understanding of the algorithm the following notations/simplifications are adopted:

- ε represents an empty or null element.
- For each function/mapping `F`, `F(`p`)` evaluates to ε if p evaluates to ε.
- Expression `F` $\vee$ `G` evaluates to `G` if `F` evaluates to ε, or `F` otherwise.
- The comparison operators ($=$, $\neq$, $>$, $\geq$, $\leq$, $<$) between a pair of nodes refer to their order in a tree (e.g., $p < q$ asserts that p is before q in the tree),
- `Head(`G`)` denotes the head of a list for which `NextPreorder` returns the root node of G.

```
input  : pattern node v
         target node w_start
         partial map f
output: target node found or ε if not found
1  if v is not the list head then
2      w_anc := f(Parent(v)) ∨ w_start
3      if Visited(v,w_start,w_anc) has not been initialized then
4          w := Visited(v,w_start,w_anc)
5      else
6          w := FindPartialCandidate(v, w_start, w_anc)
7          Visited(v,w_start,w_anc) := w
8      if w ≠ ε then
9          f(v) := w                                          ◄ mapping
10     else
11         return ε
12 v_next := NextPreorder(v)
13 continue := true
14 while v_next ≠ ε and continue = true do
15     w_anc := f(Parent(v_next)) ∨ w_start
16     v_sibling := PreviousSibling(v_next)
17     while v_sibling ≠ ε and f(v_sibling) = ε do
18         v_sibling := PreviousSibling(v_sibling)
19     w_start_next := GetFirstPotentialCandidate(v_next, w_anc, f(v_sibling))
20     total_maps := |Maps|
21     while w_start_next ≠ ε do
22         w := FindMatch(v_next, w_start_next, f)             ◄ recursion
23         if w ≠ ε then
24             w_start_next := GetNextPotentialCandidate(v_next, w_anc, w)
25         else
26             w_start_next := ε
27     if v_next is optional and |Maps| = total_maps then
28         v_next := NextPreorder(LastLeaf(v_next))
29     else
30         continue := false
31 if v_next = ε then                                         ◄ solution
32     add f to Maps
33 return f(v)
```

Figure 4.3: Pseudocode of the backtracking algorithm `FindMatch`

```
input  : pattern node v
         target node to start with w_start
         ancestor target node w_anc
output: target node w or ε if not found
1  w := w_start
2  repeat
3  |  if v matches the label of w
4  |  and v matches the content of w if there is any
5  |  and w satisfies v's ancestor path constraints if there are any
6  |  and w satisfies v's sibling path constraints if there are any
7  |  then return w                                              ◄ found
8  |  w := GetNextPotentialCandidate(v, w_anc, w)
9  until w ≠ ε and w ≤ LastLeaf(w_anc)
10 return ε                                                     ◄ not found
```

Figure 4.4: Pseudocode of the `FindPartialCandidate` function

```
input  : pattern node v
         ancestor target node w_anc
         target node w_prev mapped to v's previous sibling (if any)
output: target node w or ε if not found
1  if w_prev ≠ ε then
2  |  if v has a child-of constraint then
3  |  |  w := FirstChild(w_anc)
4  |  |  repeat
5  |  |  |  w := NextSibling(w)
6  |  |  until w ≠ ε and w ≤ w_prev
7  |  else if v has a sibling constraint then
8  |  |  w := NextSibling(w_prev)
9  |  else
10 |  |  if LastLeaf(w_prev) < LastLeaf(w_anc) then
11 |  |  |  w := NextPreorder(LastLeaf(w_prev))
12 |  |  else
13 |  |  |  w := ε
14 else
15 |  w := FirstChild(w_anc)
16 return w
```

Figure 4.5: Pseudocode of the `GetFirstPotentialCandidate` function

```
input  : pattern node v
         ancestor target node w_anc
         target node w_prev mapped to v
output: target node w or ε if not found
1 if v has a child-of or sibling constraint then
2 |  w := NextSibling(w_prev)
3 else
4 |  if w_prev < LastLeaf(w_anc) then
5 |  |  w := NextPreorder(w_prev)
6 |  else
7 |  |  w := ε
8 return w
```

Figure 4.6: Pseudocode of the `GetNextPotentialCandidate` function

The general idea of the backtracking algorithm `FindMatch` is to incrementally construct a *partial map*, extending it by adding *partial candidates* and abandoning this process (as early as possible) if it becomes apparent that the map cannot result in a *complete map*, which signals that further exploration be of no avail.

Definition 4.4.2 (Partial map) *Given a pattern tree P and a target tree T, a* partial map *f_n is a map from $P_{\leq n}$ to T, where $P_{\leq n}$ is a subtree of P consisting of n first nodes of P and $n \leq |P|$, such that f_n satisfies the conditions of the constrained constituent tree inclusion problem.*

Definition 4.4.3 (Complete map) *Given a pattern tree P and a target tree T, a partial map f_n is* complete *if $n = |P|$.*

Definition 4.4.4 (Partial candidate) *Given a pattern tree P, a target tree T, and a partial map f_n, a* partial candidate *is any node $w \in T$ such that $\exists v \in P : f_n(v) = w$.*

Definition 4.4.5 (Potential candidate) *Given a pattern tree P, a target tree T, and a partial map f_n, a* potential candidate *is a node $w \in T$ such that w satisfies the* tree structure *conditions for $v_{n+1} \in P$, where v_{n+1} is the next target node for which a counterpart node (partial candidate) is being sought.*

From the above, *complete map* is also *partial map*, and *partial candidate* is also *potential candidate*. The reverse does not hold.

Calling `FindMatch(Head(`P`), Head(`T`),` f_ϕ`)`, where f_ϕ denotes an initial, empty map, produces a set of maps from P to T.

Given a pattern node v, a potential node w_{start}, and a partial map f, `FindMatch` is trying to find a match for v exploring the subtree of T beginning with w_{start} (lines 1–11). If a matching node is found, it is added to the partial map (line 9) and eventually returned (line 33). Before returning the newly mapped node, the algorithm continues building the map f by selecting the next pattern node v_{next}, and recursively and repeatedly calling itself with v_{next}, the augmented partial map f, and w_{start} recalculated with each repetition (lines 12–30). If the algorithm completes the map, i.e., $v_{next} = \varepsilon$, a solution is found and the map f is saved (lines 31–32).

Due to the fact that the string conditions (the conditions involving string comparisons) are the most time-consuming of all the conditions in Definition 4.4.1, the algorithm limits the search space in two steps. The first step takes into consideration only tree structure conditions, which serve to nominate the limit nodes in T (lines 14–30), whereas the second step proceeds with the string conditions looking for a match only within the nominated bounds (lines 1–11).

Both functions `GetFirstPotentialCandidate` and `GetNextPotentialCandidate`, presented in detail in Figures 4.5 and 4.6, respectively, are guaranteed to return a potential node, which satisfies the tree structure conditions, or ε if such a node does not exist, for the current pattern node with the current partial map. Subsequently, given the potential node w_{start}, the `FindPartialCandidate` function, presented in Figure 4.4, is guaranteed to return a partial node, which satisfies the remaining conditions, or ε if such a node does not exist. `FindPartialCandidate` includes calls to `GetNextPotentialCandidate` itself.

The optional node resolution is embedded in the tree-structure-condition step of the algorithm (lines 14–30). If no new maps have been produced for v_{next} and if v_{next} is optional, the algorithm chooses another pattern node with which it proceeds (lines 27–28). The skipped node and its descendants are ignored wherever necessary (lines 17–18).

Exploring the many possibilities of mapping P to T often results in revisiting parts of T under similar conditions for which the outcome is exactly the same as in the previous visits. This problem is overcome by a hash function/table (caching map), `Visited`, which returns the previously found outcome and skips the costly `FindPartialCandidate` function (lines 3–4).

Correctness and completeness

In this section, the algorithm's correctness is (informally) proven by showing that the algorithm follows the conditions in Definition 4.4.1, whereas its completeness is proven by showing that it produces *all* the maps.

The order condition is taken care of by traversing the trees in preorder.

Given the partial map f and the next pattern node v_{next}, both `GetFirstPotentialCandidate` and `GetNextPotentialCandidate` return a potential candidate (as given in Definition 4.4.5) if there is any, which is shown below.

The ancestor–descendant condition is satisfied as long as the candidate node is included in the subtree of $f(\texttt{Parent}(v_{next}))$ and, in case v_{next} has a previous sibling, the candidate is after (has the preorder number greater than) the last descendant of $f(\texttt{PreviousSibling}(v_{next}))$.

If v_{next} is a first child of $\texttt{Parent}(v_{next})$, i.e., v_{next} has no previous siblings, the first potential candidate is the first child of $f(\texttt{Parent}(v_{next}))$, which satisfies the ancestor–descendant condition. Otherwise (v_{next} does have a previous sibling), determining the first potential candidate depends on v_{next}'s tree structure constraints.

If v_{next} does not have any tree structure constraints, then the first potential candidate is the first node with the preorder number greater than the one of the last descendant of $f(\texttt{PreviousSibling}(v_{next}))$. The first potential candidate does not exist in that situation if the last descendant of $f(\texttt{PreviousSibling}(v_{next}))$ is at the same time the last descendant of $f(\texttt{Parent}(v_{next}))$, which satisfies the ancestor–descendant condition.

If v_{next} has a sibling constraint then the first potential candidate is the next sibling, if any, of $f(\texttt{PreviousSibling}(v_{next}))$. Finally, if v_{next} has a child-of constraint then the first potential candidate is the first child of $f(\texttt{Parent}(v_{next}))$ with the preorder number greater then that of $f(\texttt{PreviousSibling}(v_{next}))$, if there is any. Both constraint conditions also naturally satisfy the ancestor–descendant condition.

All of the above cases are comprised by the `GetFirstPotentialCandidate` function, thus `GetFirstPotentialCandidate` returns a potential candidate if there is any. □

Given a potential candidate for v_{next}, the next potential candidate for v_{next} is the next sibling, if any, of the previous potential candidate if v_{next} has any kind of tree structure constrained; otherwise, the next node in preorder. In the latter case the next potential candidate does not exist if the previous potential candidate is the last descendant of $f(\texttt{Parent}(v_{next}))$.

The two aforementioned cases are comprised by the `GetNextPotentialCandidate` function, thus `GetFirstPotentialCandidate` returns a potential candidate if there is any. □

Showing that `FindPartialCandidate` returns a partial candidate is straightforward. The function checks if a potential candidate (returned either by `GetFirstPotentialCandidate` or `GetNextPotentialCandidate`) satisfies the string constraint conditions and returns the potential candidate if it does or continues by checking the next potential candidates. If a potential candidate satisfies the string constraint conditions then a potential candidate is also a

partial candidate, thus `GetFirstPotentialCandidate` returns a partial candidate if there is any. □

A partial map f_n is built up by recursively calling `FindMatch`. There are two halting points in the recursion: (1) if a partial candidate cannot be found, and (2) if the partial map is complete, i.e., $n = |P|$. Only the second case produces a complete map satisfying the conditions as given in Definition 4.4.1, thus the `FindMatch` algorithm is correct. □

Given the current state of a partial map, the next partial candidate is sought repeatedly for each potential candidate. The repetition continues regardless of whether the found partial candidates eventually lead to a solution or not. This procedure is repeated recursively for every state of the partial map, resulting in it finding *all* possible maps from P to T. Thus, the `FindMatch` algorithm is complete. □

Complexity

The tree inclusion algorithms have been shown to have the time and space complexity of $O(|P| \cdot |T|)$ [77, 120, 76], which is satisfied as long as an algorithm compares a pair of pattern–target nodes no more than once. In order to do that, the previously proposed algorithms use dynamic programming storing the results of subcomputations in a $|P| \times |T|$ table and using it whenever a comparison is repeated for the same pair of nodes. In the constrained constituent tree inclusion algorithm a similar table (`Visited` in Figure 4.3) is used. However, due to the ancestor path pattern constraint, the table must also store a target ancestor node for a given pair of pattern–target nodes. This subsequently implies that a pattern–target node pair can be visited more than once with a different target ancestor. The number of target ancestor nodes for a given pair is limited by the depth of a target tree T. The rough upper noninclusive bound of the total number of comparisons is therefore $|T| + (|P| - 1) \cdot |T| \cdot \sum_{i=2}^{Depth(T)} (i-1)p_i$, where p_i is the normalized number of target nodes on i-th level, such that $\sum_{i=1}^{Depth(T)} p_i = 1$, which in turn is bound by $O(|P| \cdot |T| \cdot Depth(T))$.

An empirical insight into the time and space complexity is given in Section 4.5.5.

4.4.3 Patterns

In this section a syntax of pattern trees, which accommodates the properties specified in Section 4.4.1, is proposed. The syntax is based on the bracketed constituent tree syntax and its grammar is shown in Figure 4.7.

The node definition starts with a label optionally followed by a list of constraints. Each constraint starts with an '@' symbol and is followed by the acronym of the constraint (e.g., `nsp` stands for *next sibling pattern*). In the case of string constraints, there are two comparison

```
pattern     = nonterminal | terminal ;

nonterminal = {gap} , "[" , node-def, [constraints] , (nonterminal | terminal) , "]" , {gap} ;
terminal    = {gap} , "[" , node-def, [constraints] , [gap , content] , "]" , {gap}

node-def    = ( "(" , label , ")" ) | label  , ["?"] ;
constraints = [constraint-sibling] , [constraint-child-of | constraint-ancestor-path] ,
              [constraint-previous-sibling-path] , [constraint-next-sibling-path] ;

constraint-child-of               = "@c" ;
constraint-sibling                = "@s" ;
constraint-ancestor-path          = "@ap" , comparison-operator , path ;
constraint-previous-sibling-path  = "@psp" , comparison-operator , path ;
constraint-next-sibling-path      = "@nsp" , comparison-operator , path ;

comparison-operator = "=" | "!="

label   = string | regex ;
path    = string | regex ;
content = string | regex ;

gap   = space | tabulation | new-line ;
```

Figure 4.7: The syntax of pattern trees in the extended Backus-Naur form (EBNF).

operators to choose from: '=' (*equal to*) or '!=' (*different from*). Although the pseudocode of the algorithm in Section 4.4.2 does not specify the syntax of ancestor and sibling paths, it is assumed that a path is given as a string formed by concatenating target tree node labels separated by some symbol (in the implementation we adopted the '->' symbol). For instance, following the example in Figure 4.1, the path between $\texttt{ENT}_{(15)}$ and $\texttt{NP}_{(7)}$ (excluding these nodes) is expressed as `NP->VP`, hence the constraint `@ap!=/NP/` in the pattern in Figure 4.2(c). A question mark ('?') following the label is used to denote an optional node.

Additionally, labels can be enclosed in square brackets, which signals that the node is a *capture node*. The notion of capture nodes was introduced to ease the formatting of relations from inclusion maps produced by the algorithm (see *relation formatting* in Section 4.4.4 for details). The algorithm itself is indifferent to the fact that the nodes are being *captured*.

Using the bracketed constituent tree syntax as the foundation syntax is motivated by its portability, since it can be managed or visualized with the same tools for managing or visualizing constituent trees. The main drawback to this approach lies in readability, which may decrease significantly if a pattern incorporates multiple constraints and uses elaborated regular expressions in a single node.

Example pattern trees are given in Figure 4.8. The four patterns shown were developed to extract entity relations encompassing the following categories of relations:

X verb Y Entity *X* followed by a verb or a verbal adjective, followed by entity *Y*, e.g., *<transcription factor> controls <cell death>*

X not verb Y as above but with negation, e.g., *<hypoxia> not induce <ICAM-1>*

X verb prep Y similar to "X verb Y" but with a preposition before the second entity, e.g., *<I kappa B alpha> modified in <LMP-1-expressing B cells>*

X not verb prep Y as above with negation, e.g., *<NF-AT> not induced by <IL-2 stimulation>*

X verb to verb Y *X* followed by a verb or a verbal adjective, followed by an infinitive form of a verb, followed by *Y*, e.g., *<T3SO4> failed to displace <[125I]T3>*

X not verb to verb Y as above with negation

X verb to verb prep Y similar to "X verb to verb Y" but with a preposition before the second entity, e.g., *<IL-5 gene segment> sufficient to respond to <activating signal>*

X not verb to verb prep Y as above with negation

```
[/.*/
  [/NP|PRN/@c
    [(ENT)@ap!=/PP|SBAR|VP/@nsp!=/^ENT/]
  ]
  [/^VP$|^ADJP$/@ap!=/PP|NP|VP/
    [(RB)? not]
    [(/^VB|^JJ/)@ap!=/PP|NP|S/]
    [/NP|PRN/@s
      [(ENT)@ap!=/PP|SBAR|VP/@nsp!=/^ENT/]
    ]]]
```

(a)

```
[/.*/
  [/NP|PRN/@c
    [(ENT)@ap!=/PP|SBAR|VP/@nsp!=/^ENT/]
  ]
  [/^VP$|^ADJP$/@ap!=/PP|NP|VP/
    [(RB)? not]
    [(/^VB|^JJ/)@ap!=/PP|NP|S/]
    [PP@s
      [(/IN|TO/)@ap!=/NP|VP|ADJP|SBAR/]
      [/NP|PRN/@s
        [(ENT)@ap!=/PP|SBAR|VP/@nsp!=/^ENT/]
      ]]]]
```

(b)

```
[/.*/
  [/NP|PRN/@c
    [(ENT)@ap!=/PP|SBAR|VP/@nsp!=/^ENT/]
  ]
  [/VP|ADJP/@ap!=/PP|NP|VP/
    [(RB)? not]
    [(/VB|JJ|IN/)@ap!=/PP|NP|S/]
    [S@s
      [VP@c
        [(TO)]
        [VP@c
          [(VB)@c]
          [NP@ap!=/VP|ADJP|PP|SBAR/
            [(ENT)@ap!=/PP|SBAR|VP/@nsp!=/^ENT/]
          ]]]]]]
```

(c)

```
[/.*/
  [/NP|PRN/@c
    [(ENT)@ap!=/PP|SBAR|VP/@nsp!=/^ENT/]
  ]
  [/VP|ADJP/@ap!=/PP|NP|VP/
    [(RB)? not]
    [(/VB|JJ|IN/)@ap!=/PP|NP|S/]
    [S@s
      [VP@c
        [(TO)]
        [VP@c
          [(VB)@c]
          [PP@ap!=/NP|VP|ADJP|SBAR/
            [(/IN|TO/)]
            [NP@ap!=/VP|ADJP|PP|SBAR/
              [(ENT)@ap!=/PP|SBAR|VP/@nsp!=/^ENT/]
            ]]]]]]]
```

(d)

Figure 4.8: Pattern constituent trees that extract relations of the following categories: (a) "X verb Y" and "X not verb Y", (b) "X verb prep Y" and "X not verb prep Y", (c) "X verb to verb Y" and "X not verb to verb Y", and (d) "X verb to verb prep Y" and "X not verb to verb prep Y"

The patterns presented in Figure 4.8 were developed by 1) building "frames" of pattern trees that consisted of labels only, and 2) narrowing down the "greediness" of the patterns by incorporating constraints. Whereas the first step was assisted by the Penn Treebank guideline [26], the second step was the result of analysis of the most frequent structures of constituent trees that appear in a biomedical corpus. In a sense, this process is similar to

obtaining the constrained constituent pattern tree with regular expressions as presented in Figure 4.2(c) from the inclusion trees presented in Figures 4.2(a) and (b).

For instance, `[(ENT)@ap!=/PP|SBAR|VP/@nsp!=/^ENT/]` (the most common pattern node, which appears twice in each pattern tree in Figure 4.8) reads as follows:

- The label of the node must be `ENT` (not part of the Penn Treebank annotation; see Figure 4.1), i.e., an explicitly annotated entity.
- The *ancestor path* (denoted by `@ap`) must not contain nodes with labels `PP`, `SBAR`, and `VP` to ensure that the to-be-matched node (an annotated entity in this case) is not part of a preposition phrase, nor subordinate phrase, nor verb phrase, respectively. Any other labels are permitted.
- The *next sibling path* (denoted by `@nsp`) must not start with a node labeled `ENT`. This is to ensure that the to-be-matched node, an annotated entity, is not just a noun modifier of another entity, i.e., it does not stand to the left of another annotated entity in the sentence.

The performance of the patterns on two different text corpora, as well as their correctness, completeness, and applicability are discussed in Section 4.5.

4.4.4 Processing pipeline

A processing pipeline for extracting relations in sentences is shown in Figure 4.9.

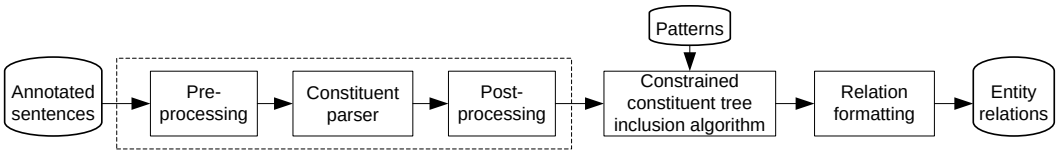


Figure 4.9: Pipeline for extracting relations in sentences.

The input sentences are assumed to have been already annotated with entities that are to be part of relations. In order to preserve the annotation and the meaning of entities (usually composed of more than one word) during parsing, it is necessary to pre-process them to the form resilient to the parser's internal tokenization process (e.g., by concatenating the words that constitute an entity with underscore characters), and to "force" the parser to treat the entities as nouns. Some parsers, such as the Stanford parser, allow for partially POS-tagged sentences, whereas others can be "deceived" by replacing the entities with some common nouns and bringing the original form back after parsing.

Post-processing of parsed sentences mainly involves replacing labels of annotated entities with the ENT tag or similar (as expected by patterns).

The constituent trees serve as input of the algorithm for solving the problem of constrained constituent tree inclusion, which produces a set of maps reflecting the patterns in the target trees. The relation formatting step involves converting the mapped nodes to relations consisting of words only, which are embedded in the nodes. Since only the leaf nodes carry the words, it is not unusual (especially with "loose" patterns) that the same set of leaves is part of two or more maps, which results in duplicating the relations. Simple string comparison statements alleviate this problem.

4.5 Evaluation

Three different types of evaluation have been performed to test for the quantitative and qualitative aspects of the proposed method. Section 4.5.2 describes experimental results on the GENIA corpus [79] with the patterns presented in Figure 4.8. The method is compared against two other approaches for extracting binary relations on this corpus. The empirical correctness and completeness of the patterns are analyzed in Section 4.5.3; whereas in Section 4.5.4 the applicability of the patterns to a general English corpus is examined by comparing the proposed method with the results generated using the open information extraction system reported in [22]. The empirical analysis of time and space complexity is presented in Section 4.5.5.

4.5.1 Corpus preprocessing

The GENIA corpus is preprocessed by decomposing nested tags and biological entities involving ellipses in coordinated clauses (an example is given in Figure 4.10).

```
<cons lex="(AND human_T_lymphocyte human_B_lymphocyte)" sem="(AND G#cell_type G#cell_type)">
  <cons lex="human*">human</cons>
  <cons lex="*T*">T</cons>
  and
  <cons lex="*B*">B</cons>
  <cons lex="*lymphocyte">lymphocytes</cons>
</cons>
```

(a)

```
<cons lex="human_T_lymphocyte" sem="G#cell_type">human T lymphocytes</cons>
and
<cons lex="human_B_lymphocyte" sem="G#cell_type">human B lymphocytes</cons>
</cons>
```

(b)

Figure 4.10: Example of (a) an ellipsis in the GENIA corpus and (b) its corresponding resolved form

The terms are further processed with a set of manually developed rules, an approach commonly used in biological entity extraction [59, 46, 45, 128, 14]. Processing terms with the rules involves removing unnecessary white spaces, dividing words and word sequences into separate instances, and removing the acronyms embedded in the sequence of words that represents their full form. A full list of rules with examples is given in Table 4.2.

Table 4.2: Preprocessing rules

1)	$\langle W\rangle - \langle s\rangle\langle D\rangle \rightarrow \langle W\rangle - \langle D\rangle$ HPP- 47.10 cell → HPP-47.10 cell
2)	$\langle C_1\rangle\langle D_1\rangle/\langle D_2\rangle\langle C_2\rangle \rightarrow \langle C_1\rangle\langle D_1\rangle\langle C_2\rangle, \langle C_1\rangle\langle D_2\rangle\langle C_2\rangle$ Arp2/3 complex → Arp2 complex, Arp3 complex
3)	$\langle D_1\rangle/\langle D_2\rangle\langle C\rangle \rightarrow \langle D_1\rangle\langle C\rangle, \langle D_2\rangle\langle C\rangle$
4)	$\langle C\rangle\langle D_1\rangle/\langle D_2\rangle \rightarrow \langle C\rangle\langle D_1\rangle, \langle C\rangle\langle D_2\rangle$ C3/5 → C3, C5
5)	$\langle C_1\rangle - \langle s\rangle and \langle s\rangle\langle C_2^{\bar{s}}\rangle - \langle C_3^{\bar{s}}\rangle \rightarrow \langle C_1\rangle - \langle C_3^{\bar{s}}\rangle, \langle C_2^{\bar{s}}\rangle - \langle C_3^{\bar{s}}\rangle$ biotin- and fluorescein-labeled amplicon → biotin-labeled amplicon, fluorescein-labeled amplicon
6)	$\langle C_1\rangle - \langle s\rangle or \langle s\rangle\langle C_2^{\bar{s}}\rangle - \langle C_3^{\bar{s}}\rangle \rightarrow \langle C_1\rangle - \langle C_3^{\bar{s}}\rangle, \langle C_2^{\bar{s}}\rangle - \langle C_3^{\bar{s}}\rangle$ bi- or multi-functional domain protein → bi-functional domain protein, multi-functional domain protein
7)	$\langle C_1\rangle\langle s\rangle(\langle C_2\rangle) \wedge \langle C_2\rangle \subset \langle C_1\rangle \rightarrow \langle C_1\rangle$ interleukin (IL)-1 → interleukin-1
8)	$\langle C_1\rangle\langle s\rangle[\langle C_2\rangle] \wedge \langle C_2\rangle \subset \langle C_1\rangle \rightarrow \langle C_1\rangle$ activator protein-1 [AP-1] site → activator protein-1 site
9)	$\langle s\rangle^{>1} \rightarrow \langle s\rangle$

$\langle C\rangle = \{character\}^+$
$\langle W\rangle = \{a|...|z\}^+$
$\langle D\rangle = \{0|...|9\}^+$
$\langle s\rangle$ = {white space character}
$\langle \bar{s}\rangle$ = {character different from $\langle s\rangle$}
$\langle C^{\bar{s}}\rangle = \langle \bar{s}\rangle\langle C\rangle\langle \bar{s}\rangle$
$\langle X\rangle \subset \langle Y\rangle$ indicates that Y contains all the characters from X in the same order

4.5.2 Comparison with existing methods

The experimental setup consists of a 500-sentence extract from the GENIA corpus [79] (see description in Section 2.2.1). In spite of having rich annotation, GENIA does not contain the annotation of relations between the annotated biological entities. Therefore, the relations of interest (see Section 4.4.3) have been manually extracted from a set of 500 randomly selected sentences that contained at least two annotated entities. The number of sentences chosen to perform the comparisons corresponds to the previous endeavors on extracting relations from GENIA reported in [37] and [14], and is five times bigger than the one used in [122]. Table 4.3 shows the distribution of relation categories in the corpus.

The patterns were applied on the output from two constituent parsers: the Stanford parser

Table 4.3: Frequency of different categories of relations in the 500-sentence extract from the GENIA corpus

Category	Frequency	
X verb prep Y	205	60.1%
X verb Y	107	31.4%
X verb to verb prep Y	12	3.5%
X verb to verb Y	11	3.2%
X not verb prep Y	3	0.9%
X not verb Y	3	0.9%
Total	341	100%

[82] (previously used in, e.g., [55, 68, 53, 75, 52]), and the Charniak-Lease parser [89], a biomedical-literature-targeted version of the highly accurate Charniak parser [32][4]. Both parsers come with pre-trained models learned from the Penn Treebank and are ready to parse raw (untagged) sentences. Two different parsers are used with the intention to investigate the independence of the method from a used parser.

Although the Penn Treebank allows one to specify grammatical function tags on constituents by adding a suffix such as -NOM (nominal), -DTV (dative), -LOC (locative), -DIR (direction), etc., different parsers use different subsets of these tags or do not produce any at all. Therefore this piece of information is completely disregarded, which also makes it possible to exclusively focus on the purely syntactic aspects of sentence decomposition and makes the method truly parser-independent.

In order to compare the performance of the proposed method with the one that would be achieved with a dependency parser, the Stanford dependency parser [49] was used. The parser maps the constituent trees obtained from the two constituent parsers to sets of typed dependencies. It uses a pattern search algorithm to identify head-dependent pairs in a constituent tree (generated previously by a constituent parser) and assigns one from over 50 grammatical dependencies to each such pair. The use of the Stanford dependency parser is motivated mainly by its modularity, which enables the use of different kinds of input (raw, POS-tagged, or fully parsed text) and different styles of dependency representations (basic, collapsed, propagated, or acyclic). The flexibility of the parser has been recognized, for instance, in comparison of the performance of different constituent parsers [39].

In order to make use of the generated typed dependencies to extract the entity relations of interest, a set of logic programming rules was prepared, in an approach similar to the one used in [122]. The set of rules has been manually tailored to maximize the value of F_1. An example of a rule written in Prolog is given in Figure 4.11. The complete set of rules used in the evaluation is provided in Appendix B.1.

[4]The performance of the two parsers have been studied in [39]

```
rel(X,Verb,Y) :- subject(Verb,X), object(Verb,Y), annotated(X), annotated(Y).
rel(X,Verb,Y) :- subject(Y,X), cop(Y,Verb), annotated(X), annotated(Y).
subject(A,B)  :- nsubj(A,B).
subject(A,B)  :- nsubjpass(A,B).
subject(G,X)  :- conj(Z,X), subject(G,Z).
subject(G,X)  :- appos(Z,X), subject(G,Z).
subject(A,B)  :- mod(B,A).
mod(A,B)      :- partmod(A,B).
mod(A,B)      :- amod(A,B).
object(A,B)   :- dobj(A,B).
object(A,B)   :- pobj(A,B).
object(G,Y)   :- conj(Z,Y), object(G,Z).
object(G,Y)   :- appos(Z,Y), object(G,Z).
```

Figure 4.11: Definition of the Prolog rule `rel/3` representing the relation category "X verb Y". The predicate `annotated/1` is not part of the Stanford grammatical dependency set, and is introduced to assert that its argument is an annotated entity in a sentence.

The rule `subject(G,X) :- conj(Z,X), subject(G,Z).` and the rule `object(G,Y) :- conj(Z,Y), object(G,Z).` in Figure 4.11 were introduced to propagate conjunct dependencies, i.e., dependencies where the two arguments are connected by a coordinating conjunction such as "and", "or", etc. Although the Stanford dependency parser is capable of returning propagated dependencies, this option was not used for two reasons: (1) preliminary experiments showed that the propagation was not always complete; and (2) using the propagation automatically forces certain dependencies to collapse including the `prep` dependency, which would only complicate the logic rules.

Extracting entity relations from typed dependencies using logic programming rules is analogous to extracting relations from constituent trees using the pattern trees. Therefore, generating dependencies directly from the constituent trees, as opposed to raw sentences, allows us to neglect any differences in the parsers' performance when comparing the pattern-tree approach with the dependency-rule approach.

The results are shown in Table 4.4. The names of the methods refer to their respective pipelines, e.g., *Stanford→dep→rules* is the method that uses the Stanford parser to obtain constituent trees which are further processed with the dependency parser that produces grammatical dependencies which, in turn, are processed with the logic programming rules resulting in entity relations. The name *frames* in the *chunker→frames* method refers to "flat" patterns and is introduced to avoid confusion with the patterns used with the constituent parsers. Additionally, the term *constituent-parser-based* is used to refer in general to the *Stanford→patterns* and *Charniak-Lease→patterns* methods, and analogically *dependency-parser-based* is used to refer to the *Stanford→dep→rules* and *Charniak-Lease→dep→rules* methods.

Both constituent-parser-based methods show similar overall performance in terms of F_1 set at the level of 77-78% and both outperform their dependency-parser-based counter-

Table 4.4: Performance of the extraction methods on the GENIA corpus. Precision (P), recall (R), and F_1 are reported in percentages.

Relation category	Method	P	R	F_1
All	Stanford→patterns	71.6	85.6	78.0
	Charniak-Lease→patterns	73.4	81.8	77.4
	Stanford→dep→rules	71.5	75.1	73.3
	Charniak-Lease→dep→rules	72.5	72.7	72.6
	chunker→frames	84.1	17.0	28.3
X verb prep Y	Stanford→patterns	67.1	81.5	73.6
	Charniak-Lease→patterns	69.8	80.0	74.5
	Stanford→dep→rules	70.0	76.1	72.9
	Charniak-Lease→dep→rules	68.4	71.7	70.0
	chunker→frames	85.7	08.8	15.9
X verb Y	Stanford→patterns	79.2	92.5	85.3
	Charniak-Lease→patterns	78.2	86.9	82.3
	Stanford→dep→rules	72.6	72.0	72.3
	Charniak-Lease→dep→rules	79.6	76.6	78.1
	chunker→frames	83.3	32.7	47.0
X verb to verb prep Y	Stanford→patterns	58.8	83.3	69.0
	Charniak-Lease→patterns	66.7	66.7	66.7
	Stanford→dep→rules	64.3	75.0	69.2
	Charniak-Lease→dep→rules	58.3	58.3	58.3
	chunker→frames	n/a	n/a	n/a
X verb to verb Y	Stanford→patterns	90.9	90.9	90.9
	Charniak-Lease→patterns	100.0	72.7	84.2
	Stanford→dep→rules	88.9	72.7	80.0
	Charniak-Lease→dep→rules	100.0	54.5	70.6
	chunker→frames	n/a	n/a	n/a
X not verb prep Y	Stanford→patterns	100.0	100.0	100.0
	Charniak-Lease→patterns	100.0	100.0	100.0
	Stanford→dep→rules	100.0	100.0	100.0
	Charniak-Lease→dep→rules	100.0	100.0	100.0
	chunker→frames	n/a	n/a	n/a
X not verb Y	Stanford→patterns	100.0	100.0	100.0
	Charniak-Lease→patterns	100.0	100.0	100.0
	Stanford→dep→rules	100.0	100.0	100.0
	Charniak-Lease→dep→rules	100.0	100.0	100.0
	chunker→frames	n/a	n/a	n/a

parts by almost five percentage points. The difference is mainly caused by the inability of the dependency parser to recognize a dependency, i.e., although some kind of dependency between two tokens is detected it is not specified and is marked as *dep* (dependent), which in the Stanford dependency parser taxonomy of dependencies is the uppermost label that generalizes all dependencies. The only relation category where a dependency-parser-based method performs better than both constituent-parser-based methods is "X verb to verb prep Y". However, the 0.2-percentage-point difference shown in this case between *Stanford→dep→rules* and *Stanford→patterns* is negligible and is not confirmed by their counterpart methods using the Charniak-Lease parser. A perfect 100% F1 was achieved by all the four methods with relations that involved the "not" adverb; however, the frequency of appearance of these categories of relations in the corpus was marginal.

Although the best precision was obtained with the method based on the shallow parser, its recall is nowhere near the recall of the remaining methods. This suggests that the shallow-parser method returns only a small set of, albeit mostly correct, relations. As expected, although this method is fast (shallow parsing is less complex than deep constituent parsing or dependency parsing) and easy to set up (it requires only flat chunking patterns), it is not suitable for complex sentences nor for complex relation categories.

4.5.3 Correctness and completeness of the patterns

Due to the fact that the patterns were not tested in an *ideal* environment, i.e., they were used with automatically generated error susceptible trees, the evaluation process was repeated taking into consideration only the correctly parsed sentences. More precisely, the branches of the 500 test trees that were either given a wrong POS or were incorrectly attached in the tree were marked as invalid. Subsequently, all retrieved and relevant relations affected by the marked branches were discarded. As long as marked branches did not influence a relation, the relation was kept. That implies that for a given tree, there were situations where some relations were kept and some were discarded.

As the result, the number of incorrect extractions was reduced to only two false positive cases and one false negative case for the Stanford parser and four false positive and one false negative for the Charniak-Lease parser, compared to 292 and 279 retrieved and relevant cases for the two parsers, respectively. The incorrectly retrieved cases came from the "*but not*" expression in sentences. For example, in the sentence "*[...] <aspirin> and <indomethacin>, but not <CyA>, induced <Hsp70 expression> [...]*" the incorrectly retrieved relation was *<CyA> induced <Hsp70 expression>*. The only relevant case that was not retrieved appeared in the sentence "*<Transcriptional regulation> [...] is mediated by <transcription factor> <Sp1> and <AP-2>*", where *<transcription factor>* was rejected due to the pattern's next-sibling constraint. The next-sibling constraint was introduced to

ignore noun modifiers; unfortunately, in this particular case it also ignored an appositional modifier. It may be argued whether any modifiers should be taken into considerations at all, since the problem could be solved by combining such entities into one single annotation in the first place. Although GENIA does allow for nested annotations, this strategy is inconsistent in this corpus. Methods based on dependency parsers have a potential advantage over the constituent-parser-based methods when it comes to distinguishing between different types of modifiers (as long as a typed dependency grammar is expressive enough); despite this, *proper* categorization of modifiers still remains an open problem.

The patterns presented in Figure 4.8 were tailored to the constituent trees it is possible to encounter in the corpus used in the evaluation, i.e., they do not account for all the (albeit unlikely) possibilities specified in the Penn Treebank guideline [26]. Thus, theoretically the patterns are neither correct nor complete. For instance, the optional node `[(RB)? not]`, which appears in every pattern in Figure 4.8, is used to "detect" verb negations. If there are two verbs attached to the same verb phrase that the optional node is attached to, but only one of these verbs is negated, the extraction process will result in an extra irrelevant relation. Similarly, the optional node will not be used at all if the negation appears in a contraction such as *don't* (not the case in scientific articles, where formal writing is used), which will result in missing a relevant relation. This and similar cases can be dealt with by extending or creating more patterns; though, the cost and complexity of such patterns, as well as the time needed to process them, may not always be justified by a (likely) negligible increase in precision and/or recall.

4.5.4 Application to a general English corpus

In order to examine the applicability of the proposed patterns on a non-biomedical text, additional evaluation was performed using the proposed method with the four patterns (in Figure 4.8) with a general English corpus used in the open information extraction project [22] . This corpus, as opposed to GENIA, is comprised of sentences that consists of exactly two annotated entities.

Table 4.5 shows the performance of the six methods including the two constituent-parser-based methods, two dependency-parser-based methods, as well as two methods reported by the authors of the corpus, O-CRF (open extraction with Conditional Random Fields) and O-NB (open extraction with Naïve Bayes classifier). Both O-CRF and O-NB are based on dependency relations (see Section 4.2 for details).

The results confirm that the constituent-parser-based methods outperform their dependency-rule-based counterparts yielding a 5-to-6-percentage-point difference in F_1. This time, however, the Charniak-Lease parser visibly outperformed the Stanford parser by a difference of

Table 4.5: Performance of the extraction methods on a general English corpus used in [22]. Precision (P), recall (R), and F_1 are reported in percentages. O-CRF and O-NB as reported in [22]

Relation category	Method	P	R	F_1
ALL	Stanford→patterns	94.2	63.0	75.5
	Charniak-Lease→patterns	95.3	71.5	81.7
	Stanford→dep→rules	94.0	55.3	69.6
	Charniak-Lease→dep→rules	95.2	63.4	76.1
	O-CRF	88.3	45.2	59.8
	O-NB	86.6	23.2	36.6
X verb Y	Stanford→patterns	98.3	63.1	76.9
	Charniak-Lease→patterns	95.5	71.5	81.8
	Stanford→dep→rules	98.9	52.0	68.1
	Charniak-Lease→dep→rules	96.8	66.5	78.8
	O-CRF	93.9	65.1	76.9
	O-NB	100	38.6	55.7
X verb prep Y	Stanford→patterns	90.2	61.8	73.3
	Charniak-Lease→patterns	95.5	71.9	82.0
	Stanford→dep→rules	89.8	59.5	71.6
	Charniak-Lease→dep→rules	96.2	56.2	70.9
	O-CRF	95.2	50.0	65.6
	O-NB	95.2	25.3	40.0
X verb to verb Y	Stanford→patterns	76.9	66.7	71.4
	Charniak-Lease→patterns	90.9	66.7	76.9
	Stanford→dep→rules	83.3	66.7	74.1
	Charniak-Lease→dep→rules	83.3	66.7	74.1
	O-CRF	95.7	46.8	62.9
	O-NB	100.0	25.5	40.6
X not verb Y	Stanford→patterns	100.0	100.0	100.0
	Charniak-Lease→patterns	100.0	100.0	100.0
	Stanford→dep→rules	100.0	100.0	100.0
	Charniak-Lease→dep→rules	100.0	100.0	100.0
	O-CRF	n/a	n/a	n/a
	O-NB	n/a	n/a	n/a

over six percentage points.

Both O-CRF and O-NB were inferior to all of the other methods, falling by over 20 and 45 percentage points, respectively, behind the best-performing Charniak-Lease→patterns method[5].

4.5.5 Runtime and memory consumption

This section presents an empirical evaluation of the time and space complexity described in Section 4.4.2.

The runtime complexity measured on a set of 4,000 experiments (500 GENIA sentences, 4 patterns, 2 parsers) shows that there is a significant margin between the scale of the problem measured as $|P| \cdot |T| \cdot Depth(T)$ and the actual number of node comparisons for a given pattern P and a target T. Figure 4.12(a) shows the distance between the actual number of comparisons for each experiment (shown as points) and the theoretical upper bound (shown as line $|P| \cdot |T| \cdot Depth(T)$); whereas the histogram in Figure 4.12(c) indicates that for each experiment the total number of comparisons never exceeds 10% of $|P| \cdot |T| \cdot Depth(T)$.

To investigate the difference in the runtime complexity between the proposed algorithm and ordered tree inclusion algorithms, which perform in $O(|P| \cdot |T|)$, the results were also projected in the function of $|P| \cdot |T|$, as shown in Figures 4.12(b) and (d). Only 0.9% of the total number of experiments demonstrate a number of comparisons above $|P| \cdot |T|$, indicating a close relationship of the constrained constituent tree inclusion algorithm with the ordered tree inclusion algorithms in terms of runtime complexity.

The algorithm implemented in Perl and run on a PC with a 1.8GHz processor took on average about 35ms and 14kB per experiment. These numbers do not include the time and memory needed for loading and storing input constituent trees.

4.6 Conclusions

The proposed method, which is based on a constituent parser and involves a set of hand-crafted patterns and an algorithm for solving the problem of constrained constituent tree inclusion, proved to be competitive when compared to other, more popular, attempts involving shallow-parser- and dependency-parser-based methods. The experimental results with a handful of patterns representing different categories of relations showed the superior

[5]Since the results were compared directly to those reported in [22], there is a chance of slight discrepancies in the interpretation of relevant relations in the process of evaluation, which could either increase or decrease the obtained difference. Nevertheless, it is unlikely that this divergence during the evaluation process meaningfully affects the results

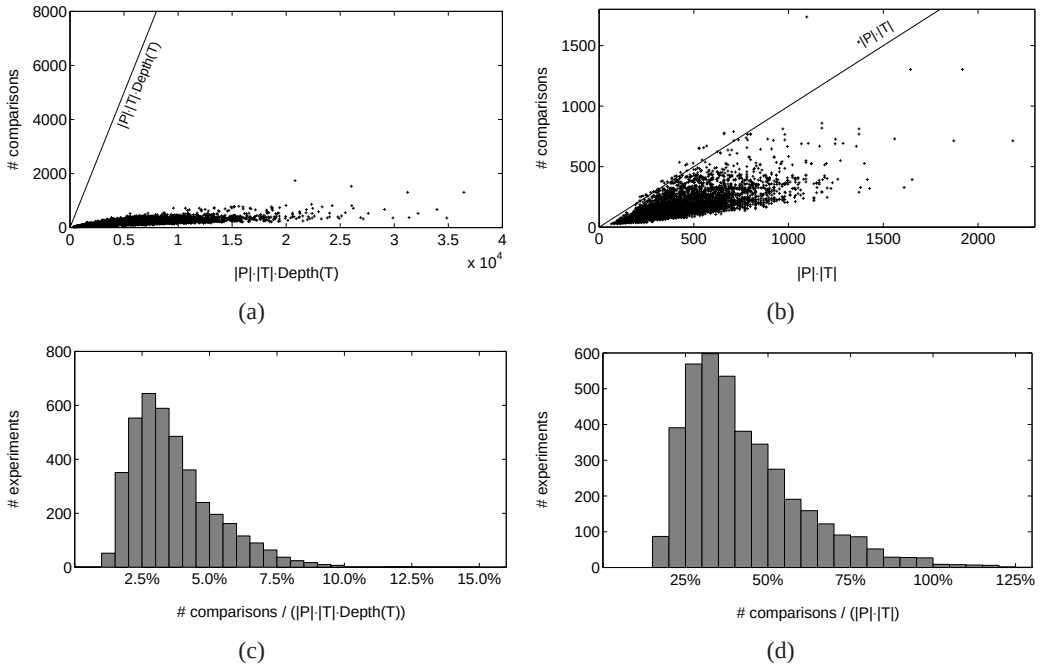


Figure 4.12: Number of node string/regex comparisons relative to the scale of the problem illustrated as scatter plots representing the 4,000 experiments in the function of (a) $|P| \cdot |T| \cdot Depth(T)$ and (b) $|P| \cdot |T|$, and their respective histograms, (c) and (d).

overall performance of the constituent-parser-based methods over the remaining methods in terms of F_1.

The expressiveness of patterns was flexible enough to cover the eight different categories of relations considered in the experimental setup with just four patterns. This was possible due to the employment of regular expressions, which allowed for defining classes of nodes and thus reducing the number of patterns, as well as the use of the tree structure and string constraints, which reduced the "greediness" of tree inclusion patterns, resulting in improved precision.

The proposed method as well as all the other discussed methods suffer from the imperfection of their corresponding parsers, which propagates through the processing pipeline and into the results. Although the proposed patterns for extracting relation categories of interest are neither fully correct nor complete, an empirical study on a subset of error-free constituent trees showed that only a small fraction (F_1 above 99%) of the total number of relations were mis-extracted.

The main advantage of the constituent-pattern-based approach is its parser independence made possible by the widely accepted Penn Treebank annotation standard; this is virtually impossible to obtain with dependency-parser-based methods. Additionally, as opposed to the dependency-parser-based methods, which require identification of grammatical dependencies/roles between words in a sentence, the proposed method proved to be a successful competitor that solely relies on the syntactic decomposition of a sentence. The main drawback of all the parser-based methods lies in their performance being undeniably dependent on the accuracy of the underlying syntactic parsers. As far as timely processing of large data is considered, the machine-learning-based method is a clear choice since, once trained, this method needs only a minimum of user input to extract relations in a single run using the output of a shallow parser. Nevertheless, the methods based on deep linguistic parsing with manually developed patterns or rules have proven to produce the most accurate results, and as such are very well suited for applications where high quality output is required.

Chapter 5

Ontology Enrichment Application

5.1 Introduction

Since its development, the GENIA corpus and ontology [79][1] have been extensively used by researchers in biological entity recognition [80, 157], ontology creation and population [128, 13], relation extraction [37, 123, 122, 14], and query answering [14, 136, 12]. These efforts show a demand for a more comprehensive and complete ontology that would incorporate extracted knowledge and semantically enhanced query answering systems. It has also been argued [130] that the structure of the ontology is inconsistent and does not fully reflect the subsumption of concepts.

This chapter discusses an extension of the GENIA ontology that not only makes the original ontology more comprehensive for reasoners but also accounts for additional information embedded in the GENIA corpus. The main focus is put on investigating the possibilities of encoding captured knowledge (as shown in the previous chapter) in OWL [6], or more specifically in OWL 2 [4], an ontology language that is becoming increasingly popular in both academic and commercial sectors [132, 54]. The proposed extension to the original ontology involves:

- asserting the category membership of biological entities,
- introducing binary relationships between biological entities,
- building the hierarchy of relationships,
- connecting the ontology with an external, well-developed source of knowledge, and

[1] See Section 2.2.1 for the description of the GENIA corpus and ontology.

- modifying and conceptually enriching the structure of the original taxonomy of categories.

The remainder of this chapter is organized as follows: Section 5.2 discusses the details of encoding the information extracted from the corpus in OWL 2. Examples of the usage of the enriched ontology are presented in Section 5.3. Section 5.4 summarizes the chapter.

A version of this chapter has been published in parts in [115, 117]

5.2 OWL representation

In the following sections, the description of encoding the knowledge base in OWL begins with the knowledge extracted from the corpus and external sources, which is then followed by discussion of the structure of the original GENIA ontology, its limitations, and the proposed re-engineering of the ontology structure.

Due to the discrepancy in naming between OWL, description logic[2], and GENIA terminology, terms such as *class*, *category*, and *concept* are used interchangeably. Similarly, the term *biological entity* is an *individual* or *instance*, and should not be confused with the term *entity* used in OWL 2 syntax to express fundamental building blocks, which include classes, individuals, properties, etc.

The OWL snippets presented in the following sections are written in the OWL *functional-style* syntax [5].

5.2.1 Biological entities

Asserting the membership of individuals (biological entities) is straightforward. Each annotated and preprocessed (see Section 4.5.1) biological entity is a member of a class indicated by the annotation. For example, in Figure 5.1, `IL-2_gene` is asserted as a member of class `DNA_domain_or_region`[3].

Due to the OWL entity naming constraints, the names of individuals are encoded (but still fully understandable by humans). For the sake of clarity each individual additionally carries a *label* property that contains the original form of the biological entity.

[2] See Section 2.1.3 for an introduction to description logic and OWL.

[3] There is some inconsistency between the corpus and the ontology in naming the categories. This issue is fixed by a simple string processor.

```
<sentence>
  <cons lex="IL-2_gene_transcription" sem="G#other_name">
    <cons lex="IL-2_gene" sem="G#DNA_domain_or_region">
      IL-2 gene
    </cons>
    transcription
  </cons>
  is affected by several
  <cons lex="nuclear_protein" sem="G#protein_family_or_group">
    nuclear proteins
  </cons>
  .
</sentence>
```

Figure 5.1: Example of an annotated sentence from the GENIA corpus. Formatting (indentation and line breaks) was added to improve readability.

Cross-referencing knowledge bases

The individuals are also associated with their definitions through UMLS Metathesaurus [9], a large vocabulary database that contains information about biomedical and health-related concepts. To properly link the biological entities with the database entries, the biological entities were furthered normalized and compared against one of the indices of the local copy of the Metathesaurus database. This process produced a *Concept Unique Identifier* (CUI) for each *recognized* biological entity. CUIs are introduced to the ontology with a functional data property, `hasCUI`, defined as follows:

```
FunctionalDataProperty(hasCUI)
DataPropertyDomain(hasCUI owl:Thing)
DataPropertyRange(hasCUI xsd:IDREF)
```

As an example, the following statements represent OWL constructs encompassing declaration, class and data property assertions, and annotation of *IL-2 gene*:

```
EntityAnnotation(Individual(IL-2_gene) Label("IL-2 gene"))
ClassAssertion(DNA_domain_or_region IL-2_gene)
DataPropertyAssertion(hasCUI IL-2_gene "C0879590")
```

Although the `hasCUI` property is rather useless when it comes to inferring facts in the ontology, it is a valuable piece of information for a user who needs to cross-reference two different knowledge bases.

Lexical nesting of biological entities

The nested tags in the corpus serve as an additional piece of information that was used to add two object properties, `lexicallyStemsFrom` and `isLexicalStemFor`. The two properties were introduced to denote one individual as lexically composed from the other.

These properties are *irreflexive object properties* in *inverse* relation to each other. An individual can stem from more than one individual and an individual can be a root for many individuals, i.e., the two properties are not *functional*. This property is defined in OWL as follows:

```
IrreflexiveObjectProperty(lexicallyStemsFrom)
IrreflexiveObjectProperty(isLexicalStemFor)
InverseObjectProperties(isLexicalStemFor lexicallyStemsFrom)
```

For example, the fact that <IL-2 gene transcription> lexically stems from <IL-2 gene> is asserted as follows:

```
ObjectPropertyAssertion(
  Comment("IL-2 gene transcription stems from IL-2 gene.")
  lexicallyStemsFrom IL-2_gene_transcription IL-2_gene)
```

The irreflexive property characteristic has rather limited inference capabilities in the ontology, and was introduced more as a constraint to prevent from mistakenly adding statements which assert that an individual stems from itself. It can also be argued whether `lexicallyStemsFrom` and `isLexicalStemFor` aim to make statements about the relationship of labels or about the relationship of the individuals they represent. Although the lexical decomposition is directly related to the labels, it is assumed that such statements have inference value for the individuals as well.

Acronyms

The identification of acronyms (see Section 2.2.1) leads to another fact that can be stated about two individuals; when one is an acronym of the other, the two are the same. For instance, knowing that *IL-2 gene* is an acronym for *interleukin-2 gene* the following is stated:

```
SameIndividuals(interleukin-2_gene IL-2_gene)
```

5.2.2 Relationships between biological entities

The TBox of the ontology is extended by declaring a set of object properties that will be used to assert verb relationships between biological entities in the corpus. These relationships are fed by the relationship extraction process as described in Chapter 4. In fact, the extracted triples are used to enrich both the TBox and the ABox by (1) deriving a hierarchical structure of object properties based on the syntax of expressions denoting kinds of

functional relationships between two entities (e.g., *activates*, *inhibits*, *is affected by*, etc.), and by (2) asserting a relationship between the *subject* and the *object* of the expression.

For instance, the relationship *<IL-2 gene transcription> is affected by <nuclear protein>* translates to:

```
ObjectPropertyAssertion(
  Comment("IL-2 gene transcription is affected by nuclear protein.")
  affectedBy IL-2_gene_transcription nuclear_protein)
```

The hierarchy of verb expressions is built by looking for expressions that have the same verb but different prepositions. For example, `affect` subsumes `affectIn`, `affectWith`, etc. An exception to this rule is made whenever the preposition *by* is encountered, which suggests that a verb expression with this preposition is in inverse relation to the verb alone. For example, `affectedBy` is the inverse of `affect`. The above is stated in OWL as follows:

```
SubObjectPropertyOf(affectIn affect)
SubObjectPropertyOf(affectWith affect)
InverseObjectProperties(affectedBy affect)
```

5.2.3 Classes

The original GENIA taxonomy consists of only the declaration of classes and axioms about subclasses. For example (translating from the original RDF/XML syntax to the functional-style syntax):

```
SubClassOf(DNA_domain_or_region DNA)
SubClassOf(DNA_family_or_group DNA)
SubClassOf(DNA_molecule DNA)
SubClassOf(DNA_substructure DNA)
SubClassOf(DNA_ETC DNA)
```

An analysis of this taxonomy and the annotation of biological entities in the corpus delivers additional, potentially useful pieces of information:

- there are a few *default* terminal classes that serve as placeholders for instances that do not belong to any of these classes' siblings,
- only terminal classes have instances, and
- an instance belongs to one and only one class.

The fact that the biological entities directly belong to the terminal classes can be embedded in the ontology by introducing *covering axioms*, such that $C \equiv D_1 \sqcup \cdots \sqcup D_n$, where

$D_1, \ldots, D_n$ are subclasses of class C, i.e., if an individual is a member of class C it must also be a member of at least one of C's subclasses. The "at least one" expression is further narrowed to "exactly one" by declaring that the sibling classes are disjointed, i.e., $D_1 \sqcap \cdots \sqcap D_n \sqsubseteq \bot$. The disjointedness of classes is assumed by the fact that each distinct biological entity that appears in the corpus is annotated to one and only one class.

The last axiom also addresses the issue of the default classes. Treating a default class in the same way as other sibling classes (as this is the case in the original GENIA ontology) does not fully reflect its meaning, namely a complement of the sibling classes. For example, in the original taxonomy the class DNA contains five subclasses out of which DNA_ETC is a subclass that represents instances that do not fall into the remaining DNA_family_or_group, DNA_domain_or_region, DNA_molecule, or DNA_substructure classes. Thus, the only way to differentiate between the default class and its siblings is the name of the class which, in the original ontology, is prefixed or suffixed by either "ETC" or "other". Although such a notation is sufficient for a human to interpret, it lacks any meaning for a reasoner.

For instance, the DNA_ETC class and its siblings can be reduced to the following expression in OWL:

```
EquivalentClasses(
  Comment("DNA_ETC is an intersection of DNA and the complement
           of DNA_ETC's siblings.")
  DNA_ETC
  ObjectIntersectionOf(
   DNA
    ObjectComplementOf(
      ObjectUnionOf(
        DNA_domain_or_region
        DNA_family_or_group
        DNA_molecule
        DNA_substructure))))
```

There are five such cases in the GENIA ontology including the three top-level classes, which suggests that Other_name is a *complement* of the *union* of Substance and Source.

Inconsistencies in the GENIA ontology

It has been argued [130] that the structure of the original GENIA ontology is inconsistent and does not fully reflect the subsumption of concepts. For instance, Protein_family_or_group appears as a subconcept of Protein, wherein the more accurate relationship between these categories would be described as: Protein *belongs to* Protein_family_or_group. Similarly, Protein *is composed of* Amino_acid rather than *is an* Amino_acid.

Such inconsistency may lead to surprising results when reasoning is applied.

To overcome this problem a structure-driven ontology evolution is proposed. Figure 5.2 shows a fragment of the original GENIA ontology and its corresponding restructured counterpart[4].

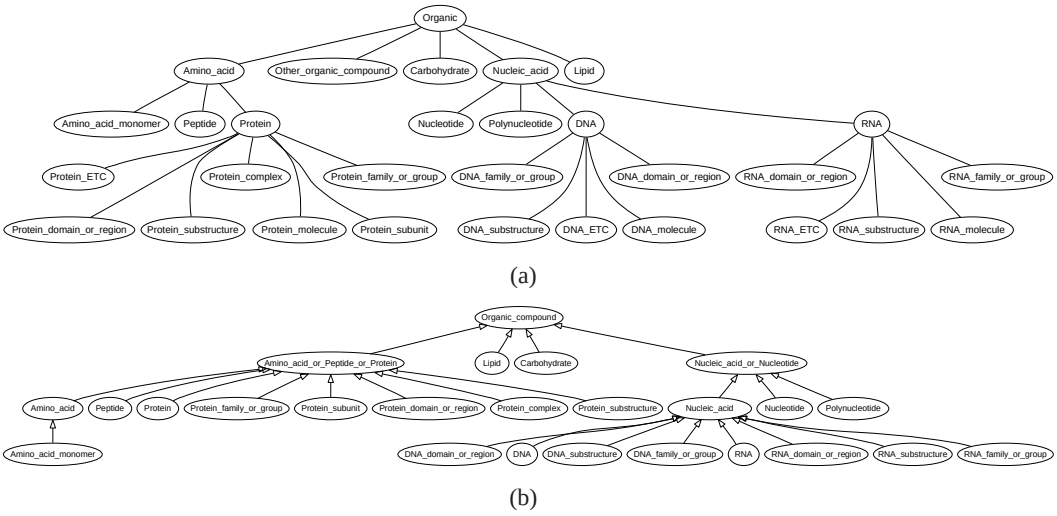


Figure 5.2: A fragment of (a) the original GENIA ontology and (b) its corresponding restructured version

The following list summarizes the structural changes of the GENIA ontology.

- The default classes have been removed. Individuals that belonged to these classes are assigned an immediately subsuming class, e.g., individuals of the former class `Protein_ETC` are moved to `Protein`. That implies that the statement $C \equiv D_1 \sqcup \cdots \sqcup D_n$, where $D_1, \ldots, D_n$ are subclasses of class C, no longer holds, i.e., it is now possible to assert a membership to any class (not only terminal).

- Similar modification was made with the classes `Protein_molecule`, `DNA_molecule`, and `RNA_molecule`. Analysis of the membership of individuals in the corpus revealed that the annotators used these classes as placeholders for `Protein`, `DNA`, and `RNA`, respectively.

- Controversial *is-a* relationships have been removed. For instance, all the former subclasses of `Protein`, except `Protein_subunit`, are now at the same level as `Protein`, and `Protein` itself is no longer a subclass of `Amino_acid`.

- In order to distinguish between organic compounds related to amino acids and nucleic acids, two "grouping" classes have been added, `Amino_acid_or_Peptide_-`

[4]The proposed modification of the GENIA ontology was undertaken in collaboration with Inge Christiaens, an expert in gene-environment interactions.

or_Protein and Nucleid_acid_or_Nucleotide. For instance, Amino_acid_or_-Peptide_or_Protein is defined as "amino acids and chains of amino acids connected by peptide linkages"[5]. Such grouping is also motivated by the fact that there are cases in the GENIA corpus where the same individuals are assigned two different classes (which shows that the annotators could not unanimously decide on a single membership). This ambiguity is resolved by asserting such individuals to a class one level higher in the hierarchy, in most cases, to the two grouping classes.

Relationships between classes

The modification of the original ontology allows for additional relationships (other than *is-a*) that can be explicitly stated in the ontology by introducing axioms using *existential quantifications*. For instance, the statement "proteins are composed of peptides, which in turn, are composed of amino acids" can be expressed as Protein $\sqsubseteq$ $\exists$composedOf.Peptide and Peptide $\sqsubseteq$ $\exists$composedOf.Amino_acid, where the composedOf role is a *transitive* object property, or in OWL:

```
TransitiveObjectProperty(composedOf)
SubClassOf(Protein ObjectSomeValuesFrom(composedOf Peptide))
SubClassOf(Peptide ObjectSomeValuesFrom(composedOf Amino_acid))
```

The above structure-driven statements can be augmented by data-driven statements, i.e., the relationship between individuals (biological entities) can be used to show the relationships between the classes they belong to. Therefore, for each class C_D: $C_D \sqsubseteq (\exists R.C_{R1} \sqcup \cdots \sqcup \exists R.C_{Rn})$, where $C_{Ri}, \ldots, C_{Rn}$ is a set of *filler* classes obtained by looking up the membership of the right-hand-side individuals directly from all the relationship triples with the property R that have a left-hand-side individual of class C_D.

For example, the OWL construct for class DNA_domain_or_region with object property affect is encoded as follows:

```
SubClassOf(
  DNA_domain_or_region
  ObjectUnionOf(
    ObjectSomeValuesFrom(affect Cell_type)
    ObjectSomeValuesFrom(affect Protein_family_or_group)
    ...))
```

The quantification constructs discussed above appear as the *necessary* criteria for the classes, i.e., they are *subsuming* the classes. They cannot appear as the *necessary and sufficient* cri-

[5]This definition is part of UMLS Semantic Network [10].

teria for those classes, i.e., they cannot be *equivalent to* those classes. Otherwise this would result in inconsistencies. Due to class disjointedness and multiple object properties cross-referencing throughout all the classes, it can be stated that if an individual is a member of a particular class, it satisfies the necessary criteria given for that class, but the inverse does not hold. For instance, it is valid to reason that a member of the class `Protein` is also a member of things that are composed of `Amino_Acid`s, but it is invalid to state that a member of things that are composed of `Amino_Acid`s is also a `Protein`, i.e., this is not always true.

5.3 Case study

This section presents several examples of using the enriched ontology in query-answering and visualization scenarios.

Example 1: Relationships between individuals

Query: "What is the relationship between *nuclear factor-kappa B* and *nuclear factor-kappa B site*?"

Result: "1) nuclear factor-kappa B and nuclear factor-kappa B site *are* both *organic compounds*; and 2) nuclear factor-kappa B site *activates* IL-2R alpha enhancer, which *binds* nuclear factor-kappa B."

Explanation: The first part of the result is obtained by following the path in the class hierarchy tree between the immediate classes the two entities are members of and their least common subsumming class. Whereas the first part involves the use of asserted statements only, the second part uses asserted statements to infer new knowledge. The assertions, with regard to the query, include:

(a) `ObjectPropertyAssertion(bind IL_2R_alpha_enhancer NF_kappa_B);`

(b) `ObjectPropertyAssertion(activatedBy IL_2R_alpha_enhancer NF_kappa_B_site);`

(c) `SameIndividuals(nuclear_factor_kappa_B NF_kappa_B);`

(d) `SameIndividuals(nuclear_factor_kappa_B_site NF_kappa_B_site);` and

(e) `InverseObjectProperties(activatedBy activate).`

The reasoning process involves the use of (c), (d), and (e) to infer that (due to the fact that *nuclear factor kappa-B* is the same entity as *NF kappa-B*, *nuclear factor kappa-B site*

is the same entity as *NF kappa-B site*, and *activated by* is in inverse relation to *activate*) *nuclear factor kappa-B site activates IL-2R alpha enhancer* and *IL-2R alpha enhancer binds nuclear factor kappa-B.*

Example 2: Relationships between classes

Query: "What kind of concepts are *composed of amino acids*?"

Result: "*Protein complexes*, *proteins*, and *peptides* are composed of amino acids."

Explanation: This query involves the use of the following assertions:

(a) `TransitiveObjectProperty(composedOf)`;

(b) `SubClassOf(Protein_complex ObjectSomeValuesFrom(composedOf Protein))`;

(c) `SubClassOf(Protein ObjectSomeValuesFrom(composedOf Peptide))`; and

(d) `SubClassOf(Peptide ObjectSomeValuesFrom(composedOf Amino_acid))`.

Due to the chain of assertions (b)–(d) and the transitivity of the mereological *composed-of* property, it is inferred that, apart from *peptide*, *protein* and *protein complex* are also composed of amino acids.

The result of this query could also be augmented by the list of individuals that belong to the three classes, `Protein_complex`, `Protein`, and `Peptide`.

Example 3: Visual representation

Figure 5.3 is an example of the visual representation of classes, individuals, and relationships mentioned in the previous two examples.

It is important to note that the object properties, `composedOf`, `bind`, `activate`, and `activatedBy`, are obtained from the subsumption statements with existential quantifications (as explained in the previous section), and as such they do *not* imply that *all* the individuals of one class appear in a particular relationship with *all* the individuals of another class; they merely indicate the *existence* of the relationships between individuals in these classes.

5.4 Conclusions

The expressiveness of OWL 2 is well-suited to accommodate the presented knowledge base automatically extracted from a text corpus. The OWL representation encompassing the

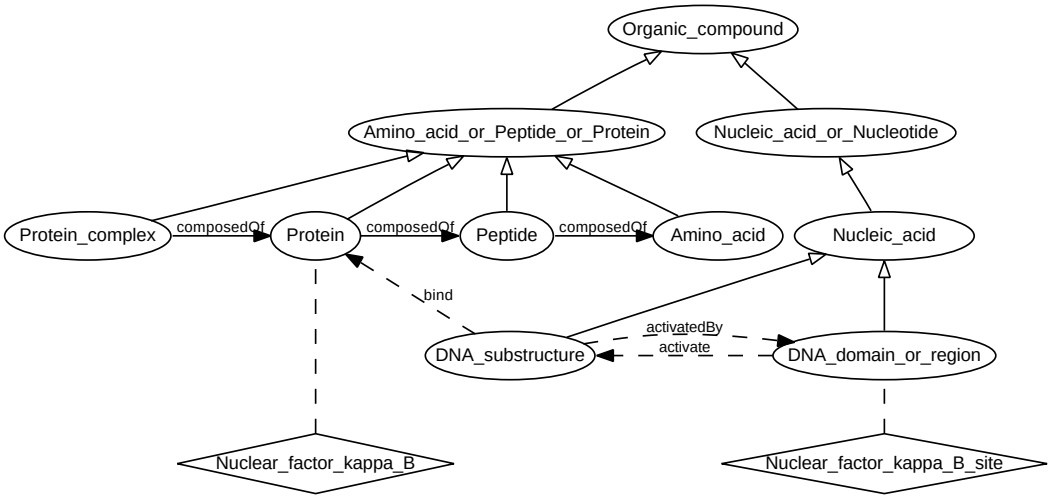


Figure 5.3: Fragment of the enriched GENIA ontology with selected individuals and relationships. Ellipses and diamonds represent classes and individuals, respectively. Solid lines indicate structure-driven relationships, whereas dashed lines indicate data-driven relationships.

knowledge base included TBox elements such as classes, data and object properties, hierarchies of classes and object properties, object properties' characteristics, as well as ABox assertions. Additionally, the structural modification of the original ontology was presented to conform with OWL and its expressive description logic. As a result, the enriched ontology became a comprehensible, inference-consistent knowledge base, which constitutes a tangible representation of knowledge and can be used in a variety of applications, such as semantically enhanced query answering systems.

Chapter 6

Conclusions

6.1 Summary

This dissertation tackled two challenging tasks in a broad textual information retrieval and knowledge capture scenario, namely text categorization and entity relation extraction. The two investigations resulted in a fast and relatively effective tool for building descriptive models capable of categorizing documents to multiple labels, and a highly effective method able to extract a broad range of relations between entities embedded in text. Additionally, an application that aims at representing the extracted knowledge in a strictly defined but highly expressive structure of ontology was presented. Such an ontology provides a tangible representation of the knowledge base and could be used to build semantically enhanced query answering systems.

The multi-label classification of documents was based on the simple and intuitive idea of building association rules with the left-hand side consisting of frequent patterns (set of words) that appear in documents, and the right-hand side composed of classes to which these patterns should be assigned. The learning process, i.e., the process of building the models, which consist of association rules, was based on a tree enumeration technique with projected datasets. The resulting algorithm allowed for the choice between two different tree traversing strategies, breadth-first and depth-first. The classification scenario involved the use of two alternative thresholding strategies based on either the document-independent confidence of the rules, or the more commonly used, cosine measure, reflecting the similarity between a rule and a document in their vector-space representations. The two introduced strategies, RCut.% and SCut.global, were used to decide on the number and selection of classes assigned to a document by choosing a fraction of top scored rules (RCut.%) or the rules that satisfied the minimum score (SCut.global). The evaluation included both the quantitative (runtime and memory consumption) and qualitative (classification effective-

ness) aspects of the proposed solution.

The extraction of relations between entities embedded in text involved the utilization of the output of a constituent parser with a set of manually developed patterns and an algorithm for extracting the patterns from the constituent trees. The algorithm was based on the ordered tree inclusion technique and was extended to solved the newly formulated problem of constrained constituent tree inclusion with regular expression matching. The proposed syntax for encoding patterns was based on the bracketed-style representation of a constituent tree. An evaluation of the method was performed with four patterns, which covered twice as many different types of relations. The constituent-parser-based approach was compared against the dependency-parser-, shallow-parser-, and machine-learning-based methods on two data sources coming from two different domains.

The extracted knowledge was further embedded into an existing ontology using OWL 2. The resulting ontology included the category membership of the entities, binary relationships between biological entities, the hierarchy of relationships, as well as cross-references between concepts in ontology and an external thesaurus. The structure- and data-driven analysis also resulted in the structural modification of the original ontology.

The major contributions of this dissertation include:

- the development of a novel, association-classification-based method for the categorization of multi-label text documents,
- a novel, tree-structure-based technique for the enumeration of frequent patterns (ruleitems) appearing in documents,
- a comparison of the depth-first and breadth-first traversals of a ruleitem tree,
- a comparison of recurrent-item and non-recurrent-item multi-label associative classification,
- a comparison of different thresholding strategies, RCut, SCut.global, and their variations, in the multi-label classification task,
- an algorithm for solving the newly-defined problem of constituent tree inclusion to extract relations between multi-word expressions (entities) embedded in text,
- the development of compact constituent-tree patterns used in the process of relation extraction,
- an evaluation of the constituent-tree patterns in the context of different constituent parsers as well as domain-orthogonal text corpora,

- a comparison of constituent-parser-based, dependency-parser-based, and machine-learning-based methods for extracting relations,
- OWL-driven enhancement of the GENIA ontology, and
- structural modification of the original taxonomy of concept in the GENIA ontology.

The list below outlines the most important findings of the aforementioned studies:

Multi-label associative classification

The tree-based enumeration of itemsets proved to be a representation perfectly suited for the incorporation of class labels and recurrent items. A set of small modifications was needed to transform the basic frequent pattern algorithm into a tool for generating multi-label, recurrent-item, classification rules.

A novel technique of storing nodes in a ruleitem tree as well as maintaining projected datasets at every node in the tree significantly reduced the number of computationally expensive candidate tests. Instead of testing each candidate against every transaction, the new technique performs this test by combining all candidate nodes into a single itemset, and therefore tests the entire set of candidates in just a single comparison.

The depth-first strategy of traversing a rulitem tree exhibits better balance between the runtime and memory consumption than the breadth-first. Although the depth-first search requires marginally more time to traverse the tree, it requires significantly less space to store the tree with the projected datasets.

The inclusion of information about the recurrence of items in transactions improved the effectiveness of classification. The difference was especially visible with macro-averaged F_1 suggesting that the recurrent-item representation works well with classes of varying distributions.

An associative-classification model incorporating multiple classes and recurrent items is faster to build than the more effective binary classifiers. The binary classifiers require a time-consuming adaptation to perform in a multi-label classification task, and therefore are ill-fitted to problems with a large number of classes.

Association rules used in associative classification models constitute a concise summary of the documents they appear in, as opposed to instance-based methods, and therefore are easier to understand and analyze.

The RCut.% strategy used with the associative classifier proved superior to the RCut strategy in the classification task that involves assinging a variable number of classes to a

document. Due to the fact that only a limited (and variable) number of rules matches a document, choosing only a (constant) fraction of the matching rules (as opposed to a fixed number of classes as is the case with RCut) is indicative of the true number of classes required by the document.

Entity relationship extraction

Extracting relations based purely on the syntactic decomposition of sentences proved to be more effective than using the output of a dependency parser. It was also shown that the presented method outperforms machine learning techniques by a considerable margin.

Pattern-based methods that extract relations from shallow phrase structure parsers, although efficient, were not suitable for a broader range of relation types.

The introduced constituent tree inclusion problem and the algorithm proposed for solving this problem waived the limitations of previously proposed algorithms for solving the ordered tree inclusion problem. This was achieved mainly due to the introduction of regular expressions and a set of optional constraints.

While being more flexible and expressive than the ordered tree inclusion algorithms, the constrained constituent tree inclusion algorithm demonstrated comparable computational complexity when compared to the former algorithms.

Regular expressions, extensively utilized in many areas of textual information retrieval and capturing (including text categorization and relation extraction), offered an indispensable mean of combining Penn Treebank tags, which enabled significant simplification of the relation extraction pattern trees.

As opposed to the dependency-parser-based methods, the proposed constituent-parser-based relation extraction was shown to be parser independent, thanks to the use of the Penn Treebank annotation standard.

The four relation extraction patterns, initially targeted towards biomedical text, performed equally well on a general English text while outperforming other methods. This suggests that the created patterns are domain independent.

Ontology enrichment

OWL was shown to be a highly expressive ontology language capable of incorporating automatically extracted knowledge in a comprehensible, reasoner-oriented fashion.

The analysis of the original structure revealed that an ontology that does not embed strict and formal definitions may be harmful to the process of reasoning and may result in false or misleading statements. The OWL's underlying description logic "forced" the structural modification of an existing ontology to improve the inference capabilities and to eliminate ambiguity or misinterpretation of the concepts stated in the original ontology.

6.2 Limitations and future directions

The studies undertaken in this dissertation constitute a few components of the bigger scenario drawn in Introduction, and thus, the scenario itself naturally dictates the direction of future development. One interesting avenue would be to explore the possibilities of using richly populated ontologies (achieved in the process of knowledge capturing) in query-answering systems, which would certainly contribute to the problem of translating a user-specified queries into more elaborate semantically enriched statements involving various relationships between the concepts in the query.

Below limitations and future directions related specifically to the work presented in this dissertation are addressed.

Multi-label associative classification

Although the proposed classifier embeds the information about the recurrence of items in transactions, this information is limited to a particular transaction only, i.e., it does not consider the frequency of words in the global document context, as it is the case with, e.g., TF/IDF. Another related problem is the lack of transaction size normalization. Although it is not a crucial step when working with the abstracts of articles, as they are usually approximately of the same size, it becomes a problem when full articles are considered.

Another possible improvement of associative classification involves the preprocessing of documents with a shallow phrase structure parser to identify noun phrases, which may be indicative of multi-word entities found in lexicons. This step would transform bag-of-words-like transactions into more meaningful sets that introduce a sense of semantics.

Entity relationship extraction

In order to maintain the community standard, the proposed syntax of patterns was based on the bracketed syntax of constituent trees. However, the readability of patterns may decrease significantly if a pattern incorporates multiple constraints and uses elaborated regular expressions; this may pose a challenge in making changes or finding potential errors. Moreover, different patterns use similar blocks of nodes, which introduces inconvenient and possibly error-prone redundancy. To reduce these problems, the development of a "metasyntax" for patterns could be considered. The metasyntax would consist of the definitions of common tree structures involving labels and/or constraints that could be grouped under easy-to-read and easy-to-interpret statements.

Another limitation comes from the fact that the patterns require the input of an expert user. In order to make the method available for end users, the development of a semi-automated tool for building constituent pattern trees is considered. The tool is expected to take a handful of relations from sentences, which are of interest to a non-expert user, and to learn the patterns based on the constituent trees that contain these relations. The automatic part of the tool involves (1) identifying common tag-blind subtrees in the constituent trees, and (2) employing the set of constraints (described in Section 4.4.1) and regular expressions to refine the patterns. The first step aims at maximizing the recall of the retrieved relations, whereas the second step reduces the number of this relations to improve precision.

Ontology enrichment

Future work related to embedding automatically extracted knowledge into an ontology includes the *generalization* of the discovered knowledge, which has already been attempted [37, 14, 36]; however, no formal representation has been proposed. The process of generalizing relationships is usually based on assigning some confidence, with which a particular relationship holds between two classes, based on the distribution of individuals in these classes. By knowing the confidence of a relationship, useful information can be inferred about (1) the significance of such a relationship, and (2) the probability of the event that a pair of individuals will participate in this relationship.

Although the confidence of relationships can be calculated indirectly by an application that works on top of an OWL ontology, this type of information cannot be explicitly stated in OWL (in neither of its versions) and, therefore, related reasoning is not possible. OWL 2 provides the expressiveness of the $\mathcal{SROIQ}$ DL language, and as such does not allow for any kind of uncertainty in individuals' membership, i.e., an individual (a pair of individuals) either belongs to a concept (a relationship) or not. This uncertainty can be introduced to the language by the fuzzy logic extension, originally proposed by [153]. The extension

affects Boolean operators and quantifiers whose range is changed from the two-value set $\{0, 1\}$ to the interval $[0, 1]$. The importance of fuzzy description logic has already been recognized. Major contributions in both defining the problem and proposing the OWL syntax of the fuzzy DL extension include [138, 58, 140] and more recently (including new features of OWL 2), [137, 28]. Recently, methodologies for evaluating the performance of probabilistic reasoners (in particular the probabilistic description logic P-$\mathcal{SHOQ}(D)$ [62]) have also been proposed [84] including a demonstration on biomedical data.

Bibliography

[1] *The DARPA Agent Markup Language Homepage*. http://www.daml.org.

[2] *Medical Subject Headings (MeSH) Fact Sheet*. http://www.nlm.nih.gov/pubs/factsheets/mesh.html.

[3] *MEDLINE Fact Sheet*. http://www.nlm.nih.gov/pubs/factsheets/medline.html.

[4] *OWL 2 Web Ontology Language Document Overview*. http://www.w3.org/TR/owl2-overview/.

[5] *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax*. http://www.w3.org/TR/2009/CR-owl2-syntax-20090611.

[6] *OWL Web Ontology Language Overview*. http://www.w3.org/TR/owl-features/.

[7] *RDF/XML Syntax Specification (Revised)*. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[8] *Reuters-21578 Text Categorization Collection*. http://kdd.ics.uci.edu/databases/-reuters21578/reuters21578.html.

[9] *UMLS Metathesaurus Fact Sheet*. http://www.nlm.nih.gov/pubs/factsheets/-umlsmeta.html.

[10] *UMLS Semantic Network Fact Sheet*. http://www.nlm.nih.gov/pubs/factsheets/-umlssemn.html.

[11] Abney, S. Partial parsing via finite-state cascades. *Nat. Lang. Eng.*, 2(4):337–344, 1996.

[12] Abulaish, M. and Dey, L. Biological ontology enhancement with fuzzy relations: A text-mining framework. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 379–385, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[13] Abulaish, M. and Dey, L. An ontology-based pattern mining system for extracting information from biological texts. In *Proceedings of 10th International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, pages 420–429, Regina, SK, Canada, August 2005.

[14] Abulaish, M. and Dey, L. Biological relation extraction and query answering from medline abstracts using ontology-based text mining. *Data Knowl. Eng.*, 61(2):228–262, 2007.

[15] Agarwal, R. C., Aggarwal, C. C., and Prasad, V. V. V. A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, 61(3):350–371, 2001.

[16] Agrawal, R. and Srikant, R. Fast algorithms for mining association rules. In *Proceedings of the International Conference on Very Large Data Bases*, pages 487–499, Santiago de Chile, Chile, 1994.

[17] Antonie, L. *Associative Classifiers: Improvements and Potential*. PhD thesis, University of Alberta, 2008.

[18] Aronson, A. R. Effective mapping of biomedical text to the UMLS Metathesaurus: the MetaMap program. In *Proceedings of AMIA Symposium*, pages 17–21, 2001.

[19] Aronson, A., Mork, J., Gay, C., Humphrey, S., and Rogers, W. The NLM indexing initiative's Medical Text Indexer. *Medinfo*, 11(Pt 1):268–272, 2004.

[20] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, 2003.

[21] Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., and Etzioni, O. Open information extraction from the web. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 51, pages 68–74, New York, NY, USA, 2007.

[22] Banko, M. and Etzioni, O. The tradeoffs between open and traditional relation extraction. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 28–36, Columbus, OH, USA, June 2008. Association for Computational Linguistics.

[23] Baralis, E. and Garza, P. Majority classification by means of association rules. In *Knowledge Discovery in Databases: PKDD 2003*, pages 34–46, Cavtat-Dubrovnik, Croatia, 2003.

[24] Berners-Lee, T., Hendler, J., and Lassila, O. The Semantic Web. *Scientific American*, 284(5), 2001.

[25] Berzal, F., Cubero, J.-C., A!nchez, D. S., and Serrano, J. A. M. A. ART: A hybrid classification model. *Machine Learning*, 54(1):67–92, January 2004.

[26] Bies, A., Ferguson, M., Katz, K., and MacIntyre, R. *Bracketing guidlines for Treebank II style*.

[27] Bikel, D. M. Design of a multi-lingual, parallel-processing statistical parsing engine. In *Proceedings of the second international conference on Human Language Technology Research*, pages 178–182, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[28] Bobillo, F. and Straccia, U. FuzzyDL: An expressive fuzzy description logic reasoner. In *Proceedings of the 2008 International Conference on Fuzzy Systems*, pages 923–930, Hong Kong, China, June 2008.

[29] Bunescu, R. and Mooney, R. J. Subsequence kernels for relation extraction. In *Proceedings of the 19th Conference on Neural Information Processing Systems*, Vancouver, BC, Canada, December 2005.

[30] Bunescu, R. C. and Mooney, R. J. A shortest path dependency kernel for relation extraction. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 724–731, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[31] Calvo, R. A., Lee, J.-M., and Li, X. Managing content with automatic document classification. *Journal of Digital Information*, 5(2), Jun 2004.

[32] Charniak, E. A maximum-entropy-inspired parser. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 132–139, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[33] Chen, J., Yin, J., Zhang, J., and Huang, J. Associative classification in text categorization. In *Proceedings of International Conference on Intelligent Computing, Advances in Intelligent Computing*, pages 1035–1044, Hefei, China, 2005.

[34] Chou, L.-F. Medline-based bibliometric analysis of gastroenterology journals between 2001 and 2007. *World Journal of Gastroenterology*, 15(23):2933–2939, 2009.

[35] Chowdhary, R., Zhang, J., and Liu, J. Bayesian inference of protein-protein interactions from biological literature. *Bioinformatics*, 25(12):1536–1542, 2009.

[36] Ciaramita, M., Gangemi, A., Ratsch, E., Saric, J., and Rojas, I. Unsupervised learning of semantic relations between concepts of a molecular biology ontology. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 659–664, Edinburgh, Scotland, July 30-August 5 2005.

[37] Cimiano, P., Hartung, M., and Ratsch, E. Finding the appropriate generalization level for binary relations extracted from the GENIA corpus. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, pages 161–169, Genoa, Italy, May 2006. ELRA.

[38] Cimiano, P. and Völker, J. Text2onto - a framework for ontology learning and data-driven change discovery. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'2005)*, pages 227–238, Alicante, Spain, 2005.

[39] Clegg, A. B. and Shepherd, A. J. Benchmarking natural-language parsers for biological applications using dependency graphs. *BMC Bioinformatics*, 8:24, January 2007.

[40] Cohen, W. W. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, California, USA, July 1995.

[41] Cohen, W. W. and Singer, Y. Context-sensitive learning methods for text categorization. *ACM Transactions on Information Systems*, 17(2):141–173, 1999.

[42] Cohen, W. Learning to classify English text with ILP methods. In *Advances in Inductive Logic Programming*. IOS, 1996.

[43] Collins, M. Head-driven statistical models for natural language parsing. *Comput. Linguist.*, 29(4):589–637, 2003.

[44] Cong, S., Han, J., Hoeflinger, J., and Padua, D. A sampling-based framework for parallel data mining. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 255–265, New York, NY, USA, 2005. ACM Press.

[45] Daraselia, N., Yuryev, A., Egorov, S., Mazo, I., and Ispolatov, I. Automatic extraction of gene ontology annotation and its correlation with clusters in protein networks. *BMC Bioinformatics*, 8:243, 2007.

[46] Daraselia, N., Yuryev, A., Egorov, S., Novichkova, S., Nikitin, A., and Mazo, I. Extracting human protein interactions from MEDLINE using a full-sentence parser. *Bioinformatics*, 20(5):604–611, 2004.

[47] Dasigi, V., Mann, R. C., and Protopopescu, V. A. Information fusion for text classification: an experimental comparison. *Pattern Recognition*, 34(12):2413–2425, 2001.

[48] Davies, J., Fensel, D., and van Harmelen, F. *Towards the Semantic Web: Ontology-driven Knowledge Management*. John Wiley & Sons, 2003.

[49] de Marneffe, M.-C., MacCartney, B., and Manning, C. D. Generating typed dependency parses from phrase structure parses. In *Proceedings of the IEEE / ACL 2006 Workshop on Spoken Language Technology*, Genoa, Italy, 2006. The Stanford Natural Language Processing Group.

[50] de Marneffe, M.-C. and Manning, C. D. *Stanford typed dependencies manual*, September 2008.

[51] Dekang, L. Dependency based evaluation of MINIPAR. In *Proceedings of the Workshop on the Evaluation of Parsing Systems, First International Conference on Language Resources and Evaluation*, Granada, Spain, 1998.

[52] Dey, L., Abulaish, M., Sharma, J., and Sharma, G. Text mining through entity-relationship based information extraction. In *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*, pages 177–180, Washington, DC, USA, 2007. IEEE Computer Society.

[53] Dingare, S., Nissim, M., Finkel, J., Manning, C., and Grover, C. A system for identifying named entities in biomedical text: how results from two evaluations reflect on both the system and the evaluations: Conference papers. *Comp. Funct. Genomics*, 6(1-2):77–85, 2005.

[54] Dolbear, C., Ruttenberg, A., and Sattler, U., editors. *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008)*, volume 432 of *CEUR Workshop Proceedings*, Karlsruhe, Germany, October 2009.

[55] Finkel, J., Dingare, S., Nguyen, H., Nissim, M., Sinclair, G., and Manning, C. Exploiting context for biomedical entity recognition: From syntax to the web. In *Proceedings of the Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA- 2004)*, pages 88–91, Geneva, Switzerland., 2004.

[56] Frank, E. and Witten, I. H. Generating accurate rule sets without global optimization. In *Proceedings of the 15th International Conference on Machine Learning*, pages 144–151, San Francisco, CA, USA, 1998.

[57] Gao, B., Liu, T.-Y., Feng, G., Qin, T., Cheng, Q.-S., and Ma, W.-Y. Hierarchical taxonomy preparation for text categorization using consistent bipartite spectral graph co-partitioning. *IEEE Transactions on Knowledge and Data Engineering, Special Issue on Data Preparation*, pages 41–50, 2005.

[58] Gao, M. and Liu, C. Extending OWL by fuzzy description logic. In *Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence*, pages 562–567, Washington, DC, USA, 2005. IEEE Computer Society.

[59] Gavrilis, D. and Dermatas, E. Automatic extraction of information from molecular biology scientific abstracts. In *Proceedings of International Workshop Speech and Computer (SPECOM-2003)*, Moscow, Russia, 2003.

[60] Genkin, A., Lewis, D. D., and Madigan, D. Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304, 2007.

[61] Geutner, P., Bodenhausen, U., and Waibel, A. Flexibility through incremental learning: Neural networks for text categorization. In *Proceedings of the World Congress on Neural Networks*, pages 24–27, Portland, OR, USA, 1993.

[62] Giugno, R. and Lukasiewicz, T. P-$\mathcal{SHOQ}(D)$: A probabilistic extension of $\mathcal{SHOQ}(D)$ for probabilistic ontologies in the semantic web. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, volume 2424 of *Lecture Notes in Computer Science*, pages 86–97, Cosenza, Italy, 2002.

[63] Han, J. and Pei, J. Mining frequent patterns by pattern-growth: methodology and implications. *SIGKDD Explorations Newsletter*, 2(2):14–20, 2000.

[64] Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M.-C. FreeSpan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 355–359, New York, NY, USA, 2000. ACM Press.

[65] Han, J., Pei, J., and Yin, Y. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 1–12, New York, NY, USA, 2000. ACM Press.

[66] Han, J., Pei, J., Yin, Y., and Mao, R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.

[67] Hersh, W., Buckley, C., Leone, T. J., and Hickam, D. Ohsumed: an interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in*

information retrieval, pages 192–201, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[68] Huang, Y., Lowe, H. J., Klein, D., and Cucina, R. J. Improved identification of noun phrases in clinical radiology reports using a high-performance statistical natural language parser augmented with the UMLS Specialist Lexicon. *J Am Med Inform Assoc*, 12(3):275–285, May 2005.

[69] Jang, H., Lim, J., Lim, J.-H., Park, S.-J., Lee, K.-C., and Park, S.-H. Finding the evidence for protein-protein interactions from pubmed abstracts. *Bioinformatics*, 22(14):e220–e226, 2006.

[70] Joachims, T. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of 10th European Conference on Machine Learning*, pages 137–142, Chemnitz, Germany, 1998. Springer Verlag, Heidelberg, DE.

[71] Joachims, T. *Learning to Classify Text using Support Vector Machines*. Kluwer Academic Publishers, Dordrecht, Netherlands, 2002.

[72] Joachims, T. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226, New York, NY, USA, 2006. ACM.

[73] Jones, S. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[74] Kankar, P., Adak, S., Sarkar, A., Murali, K., and Sharma, G. MedMeSH summarizer: Text mining for gene clusters. In *Proceeding of the 2nd SIAM International Conference on Data Mining*, pages 548–565, Arlington, VA, USA, 2002.

[75] Kaufmann, E., Bernstein, A., and Zumstein, R. Querix: A natural language interface to query ontologies based on clarification dialogs. In *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, pages 980–981, Athens, GA, USA, November 2006. Springer.

[76] Kilpeläinen, P. and Mannila, H. Query primitives for tree-structured data. In *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 213–225, London, UK, 1994. Springer-Verlag.

[77] Kilpelainen, P. and Mannila, H. Ordered and unordered tree inclusion. *SIAM J. Comput.*, 24(2):340–356, 1995.

[78] Kim, H. and Chen, S.-S. Associative naïve bayes classifier: Automated linking of gene ontology to MEDLINE documents. *Pattern Recognition*, 42(9):1777–1785, 2009.

[79] Kim, J.-D., Ohta, T., Tateisi, Y., and ichi Tsujii, J. GENIA corpus–a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19 Suppl 1:180–182, 2003.

[80] Kim, J.-D., Ohta, T., Tsuruoka, Y., Tateisi, Y., and Collier, N. Introduction to the bio-entity recognition task at JNLPBA. In *Proceedings of the International Workshop on Natural Language Processing in Biomedicine and its Applications*, pages 70–75, Geneva, Switzerland, 2004.

[81] Kleene, S. C. Representation of events in nerve nets and finite automata. *Automata Studies, Ann. Math Studies*, 34:3–41, 1956.

[82] Klein, D. and Manning, C. D. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 423–430, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[83] Klein, S. T. Combinatorial representation of generalized Fibonacci numbers. *Fibonacci Quarterly*, 29:124–131, 1991.

[84] Klinov, P. and Parsia, B. Optimization and evaluation of reasoning in probabilistic description logic: Towards a systematic approach. In *Proceedings of the 7th International Semantic Web Conference*, pages 213–228, Karlsruhe, Germany, 2008.

[85] Koller, D. and Sahami, M. Hierarchically classifying documents using very few words. In Fisher, D. H., editor, *Proceedings of the 14th International Conference on Machine Learning*, pages 170–178, Nashville, TN, USA, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[86] Lam, W. and Ho, C. Y. Using a generalized instance set for automatic text categorization. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 81–89, Melbourne, Australia, 1998. ACM Press.

[87] Lam, W., Ruiz, M., and Srinivasan, P. Automatic text categorization and its application to text retrieval. *IEEE Transactions on Knowledge and Data Engineering*, 11(6):865–879, 1999.

[88] Larkey, L. S. and Croft, W. B. Combining classifiers in text categorization. In Frei, H.-P., Harman, D., Schäuble, P., and Wilkinson, R., editors, *Proceedings of the 19th ACM International Conference on Research and Development in Information Retrieval*, pages 289–297, Zürich, Switzerland, 1996. ACM Press, New York, US.

[89] Lease, M. and Charniak, E. Parsing biomedical literature. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP-05)*, pages 58–69, Jeju Island, Korea, 2005.

[90] Lewis, D. D., Yang, Y., Rose, T., and Li, F. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[91] Lewis, D. D. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the 15th annual international ACM SIGIR conference on research and development in information retrieval*, pages 37–50, New York, NY, USA, 1992. ACM Press.

[92] Lewis, D. D., Schapire, R. E., Callan, J. P., and Papka, R. Training algorithms for linear text classifiers. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 298–306, Zurich, Switzerland, 1996. ACM Press.

[93] Li, W., Han, J., and Pei, J. CMAR: Accurate and efficient classification based on multiple class-association rules. In *Proceedings of the IEEE International Conference on Data Mining*, pages 369–376, San Jose, CA, USA, 2001.

[94] Li, Y. H. and Jain, A. K. Classification of text documents. *The Computer Journal*, 41(8):537–546, 1998.

[95] Liu, B., Hsu, W., and Ma, Y. Integrating classification and association rule mining. In *Knowledge Discovery and Data Mining*, pages 80–86, 1998.

[96] Liu, T.-Y., Yang, Y., Wan, H., Zeng, H.-J., Chen, Z., and Ma, W.-Y. Support vector machines classification with very large scale taxonomy. *SIGKDD Explorations, Special Issue on Text Mining and Natural Language Processing*, 2005.

[97] Liu, T.-Y., Yang, Y., Wan, H., Zhou, Q., Gao, B., Zeng, H.-J., Chen, Z., and Ma, W.-Y. An experimental study on large-scale web categorization. In *Proceedings of special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1106–1107, New York, NY, USA, 2005. ACM Press.

[98] McIlraith, S. A., Son, T. C., and Zeng, H. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[99] Mitchell, T. M. Machine learning and data mining. *Communications of the ACM*, 42(11):30–36, 1999.

[100] Miwa, M., Satre, R., Miyao, Y., and Tsujii, J. Protein-protein interaction extraction by leveraging multiple kernels and parsers. *International Journal of Medical Informatics*, 2009.

[101] Mukherjea, S., Bamba, B., and Kankar, P. Information retrieval and knowledge discovery utilizing a biomedical patent semantic web. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1099–1110, August 2005.

[102] Müller, H.-M., Rangarajan, A., Teal, T., and Sternberg, P. Textpresso for neuroscience: Searching the full text of thousands of neuroscience research papers. *Neuroinformatics*, 6(3):195–204, 2008.

[103] Müller, H.-M., Kenny, E. E., and Sternberg, P. W. Textpresso: An ontology-based information retrieval and extraction system for biological literature. *PLoS Biology*, 2(11), 2004.

[104] Ong, K., Ng, W., and Lim, E. Mining multi-level rules with recurrent items using FP'-Tree. In *Proceedings of the 3rd International Conference on Information, Communications and Signal Processing*, Singapore, October 2001.

[105] Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering*, page 215, Washington, DC, USA, 2001. IEEE Computer Society.

[106] Plikus, M., Zhang, Z., and Chuong, C.-M. Pubfocus: Semantic MEDLINE/PubMed citations analytics through integration of controlled biomedical dictionaries and ranking algorithm. *BMC Bioinformatics*, 7:424, 2006.

[107] Porter, M. F. An algorithm for suffix stripping. *Readings in information retrieval*, pages 313–316, 1997.

[108] Quinlan, J. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[109] Quinlan, J. R. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[110] Quinlan, J. R. Learning first-order definitions of functions. *Journal of Artificial Intelligence Research*, 5:139–161, 1996.

[111] Quinlan, J. R. and Cameron-Jones, R. M. Foil: A midterm report. In *Proceedings of the European Conference on Machine Learning*, pages 3–20, London, UK, 1993. Springer-Verlag.

[112] Quinlan, J. R. and Cameron-Jones, R. M. Induction of logic programs: FOIL and related systems. *New Generation Computing*, 13(3&4):287–312, 1995.

[113] Rak, R., Kurgan, L., and Reformat, M. Multi-label associative classification of medical documents from MEDLINE. In *Proceedings of the 4th International Conference on Machine Learning and Applications*, pages 177–184, Los Angeles, CA, USA, December 2005.

[114] Rak, R., Kurgan, L., and Reformat, M. Multilabel associative classification categorization of MEDLINE articles into MeSH keywords. *IEEE Engineering in Medicine and Biology Magazine*, 26(2):47–55, 2007.

[115] Rak, R., Kurgan, L., and Reformat, M. xGENIA: A comprehensive OWL ontology based on the GENIA corpus. *Bioinformation*, 1(9):360–362, 2007.

[116] Rak, R., Kurgan, L., and Reformat, M. A tree-projection-based algorithm for multi-label recurrent-item associative-classification rule generation. *Data and Knowledge Engineering*, 64(1):171–197, January 2008.

[117] Rak, R., Reformat, M., and Kurgan, L. Use of OWL 2 to facilitate a biomedical knowledge base extracted from the GENIA corpus. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (co-located with the 7th International Semantic Web Conference)*, Karlsruhe, Germany, October 2008.

[118] Rak, R., Reformat, M., and Kurgan, L. Extracting functional binary relations between annotated entities in text corpora: a constituent-parser-based approach. *Submitted for publication in Data and Knowledge Engineering*, 2009.

[119] Rak, R., Stach, W., Zaïane, O. R., and Antonie, M.-L. Considering re-occurring features in associative classifiers. In *Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'05)*, pages 240–248, Hanoi, Vietnam, May 2005.

[120] Richter, T. A new algorithm for the ordered tree inclusion problem. In Apostolico, A. and Hein, J., editors, *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching*, pages 150–166, Aarhus, Denmark, 1997. Springer-Verlag, Berlin.

[121] Rinaldi, F., Schneider, G., Kaljurand, K., Dowdall, J., Andronis, C., Persidis, A., and Konstanti, O. Mining relations in the GENIA corpus. In *Proceedings of the Second European Workshop on Data Mining and Text Mining for Bioinformatics*, pages 61–68, Pisa, Italy, September 2004.

[122] Rinaldi, F., Schneider, G., Kaljurand, K., Hess, M., Andronis, C., Konstandi, O., and Persidis, A. Mining of relations between proteins over biomedical scientific literature using a deep-linguistic approach. *Artif. Intell. Med.*, 39(2):127–136, 2007.

[123] Rinaldi, F., Schneider, G., Kaljurand, K., Hess, M., and Romacker, M. An environment for relation mining over richly annotated corpora: the case of GENIA. *BMC Bioinformatics*, 7:S3, 2006.

[124] Robertson, S. Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation*, 60:2004, 2004.

[125] Robertson, S. E. and Harding, P. Probabilistic automatic indexing by learning from human indexers. *Journal of Documentation*, 40(4):264–270, 1984.

[126] Ruch, P. Automatic assignment of biomedical categories: toward a generic approach. *Bioinformatics*, 22(6):658–664, 2006.

[127] Ruiz, M. E. and Srinivasan, P. Hierarchical text categorization using neural networks. *Information Retrieval*, 5(1):87–118, Jan 2002.

[128] SanJuan, E., Dowdall, J., Ibekwe-SanJuan, F., and Rinaldi, F. A symbolic approach to automatic multiword term structuring. *Computer Speech & Language. Special issue on Multiword Expression.*, 19(4):524–542, October 2005.

[129] Schneider, G., Rinaldi, F., and Dowdall, J. Fast, deep-linguistic statistical minimalist dependency parsing. In *Proceedings of the Workshop on Recent Advances in Dependency Grammars (COLING-2004)*, pages 33–40, Geneva, Switzerland, 2004.

[130] Schulz, S., Beisswanger, E., Wermter, J., and Hahn, U. Towards an upper-level ontology for molecular biology. In *Proceedings of AMIA Annu Symp*, pages 694–698, 2006.

[131] Sebastiani, F. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.

[132] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[133] Sleator, D. and Temperley, D. Parsing English with a link grammar. In *Proceedings of the 3rd International Workshop on Parsing Technologies*, Tilburg, Netherlands & Durbuy, Belgium, 1993.

[134] Smith, B., Ashburner, M., Rosse, C., Bard, J., Bug, W., Ceusters, W., Goldberg, L. J., Eilbeck, K., Ireland, A., Mungall, C. J., Leontis, N., Rocca-Serra, P., Ruttenberg, A., Sansone, S.-A., Scheuermann, R. H., and Shah, N. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology*, 25:1251–1255, 2007.

[135] Sohn, S., Kim, W., Comeau, D., and Wilbur, W. Optimal training sets for bayesian prediction of mesh assignment. *Journal of the American Medical Informatics Association*, 15(4):546–553, 2008.

[136] Song, Y.-I., Kim, S.-B., and Rim, H.-C. Terminology indexing and reweighting methods for biomedical text retrieval. In *Proceedings of the SIGIR'04 Workshop on Search and Discovery in Bioinformatics*, Sheffield, UK, July 2004.

[137] Stoilos, G. and Stamou, G. Extending fuzzy description logics for the semantic web. In *Proceedings of the 3rd International Workshop of OWL: Experiences and Directions*, Innsbruck, Austria, August 2007.

[138] Stoilos, G., Stamou, G., Tzouvaras, V., Pan, J. Z., and Horrocks, I. Fuzzy OWL: Uncertainty and the semantic web. In *In Proceedings of the International Workshop on OWL: Experiences and Directions*, Galway, Ireland, 2005.

[139] Stojanovic, L. *Methods and Tools for Ontology Evolution*. PhD thesis, FZI Research Center for Information Technologies at the University of Karslruhe, Germany, August 2004.

[140] Straccia, U. Towards a fuzzy description logic for the semantic web (preliminary report). In *Proceedings of The Semantic Web: Research and Applications, Second European Semantic Web Conference*, pages 167–181, Heraklion, Greece, 2005.

[141] Subramaniam, L. V., Mukherjea, S., Kankar, P., Srivastava, B., Batra, V. S., Kamesam, P. V., and Kothari, R. Information extraction from biomedical literature: methodology, evaluation and an application. In *Proceedings of the 12th International Conference on Information and Knowledge Management*, pages 410–417, New York, NY, USA, 2003. ACM Press.

[142] Thabtah, A., Cowling, P., and Peng, Y. Multiple labels associative classification. *Knowledge and Information Systems*, 9(1):109–129, 2006.

[143] Thabtah, F., Cowling, P., and Peng, Y. MCAR: multi-class classification based on association rule. In *Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications*, pages 33–39, Washington, DC, USA, 2005.

[144] Trieschnigg, D., Pezik, P., Lee, V., de Jong, F., Kraaij, W., and Rebholz-Schuhmann, D. MeSH Up: Effective MeSH text classification for improved document retrieval. *Bioinformatics*, 25(11):1412–1418, 2009.

[145] Uramoto, N., Matsuzawa, H., Nagano, T., Murakami, A., Takeuchi, H., and Takeda, K. A text-mining system for knowledge discovery from biomedical documents. *IBM Systems Journal*, 43(3):516–533, 2004.

[146] van Rijsbergen, C. J. *Information Retrieval*. Butterworths, London, second edition, 1979.

[147] Vanteru, B., Shaik, J., and Yeasin, M. Semantically linking and browsing PubMed abstracts with Gene Ontology. *BMC Genomics*, 9(suppl 1):S10, 2008.

[148] Willinsky, J. and Quint-Rapoport, M. How complementary and alternative medicine practitioners use PubMed. *Journal of Medical Internet Research*, 9(2), 2007.

[149] Yang, S., Needleman, H., and Niederman, R. A bibliometric analysis of the pediatric dental literature in medline. *Pediatric Dentistry*, 23(5):415–418, 2001.

[150] Yang, Y. An evaluation of statistical approaches to MEDLINE indexing. In *Proceedings of the Conference of the American Medical Informatics Association*, pages 358–362, Washington, DC, USA, 1996.

[151] Yang, Y. A study of thresholding strategies for text categorization. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 137–145, New York, NY, USA, 2001. ACM Press.

[152] Yang, Z., Lin, H., and Wu, B. Bioppiextractor: A protein-protein interaction extraction system for biomedical literature. *Expert Systems with Applications*, 36(2, Part 1):2228–2233, 2009.

[153] Yen, J. Generalizing term subsumption languages to fuzzy logic. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 472–477, Sydney, Australia, August 1991.

[154] Yin, X. and Han, J. CPAR: Classification based on predictive association rules. In *Proceedings of the 3rd SIAM International Conference on Data Mining (SDM'03)*, pages 331–335, San Francisco, CA, USA, May 2003.

[155] Zaïane, O. R. and Antonie, M.-L. Classifying text documents by associating terms with text categories. In *Proceedings of the 13th Australasian Database Conference*, pages 215–222, Melbourne, Victoria, Australia, 2002.

[156] Zaki, M. J., Parthasarathy, S., Ogihara, M., and Li, W. New algorithms for fast discovery of association rules. Technical report, Rochester, NY, USA, 1997.

[157] Zhou, G. Recognizing names in biomedical texts using mutual information independence model and SVM plus sigmoid. *International Journal of Medical Informatics*, 75(6):456–467, June 2006.

APPENDIX A Penn Treebank II Tags

The following is a *selected* set of tags used in the Penn Treebank annotation. The complete list with examples is provided in [26].

Clause level

S	simple declarative clause (not introduced by a subordinating conjunction)
SBAR	clause introduced by a subordinating conjunction

Phrase level

ADJP	adjective phrase
ADVP	adverb phrase
CONJP	conjunction phrase
FRAG	fragment
NP	noun phrase
PP	prepositional phrase
PRN	parenthetical
VP	verb phrase
WHADJP	adjective phrase introduced by a *wh*-word
WHAVP	adverb phrase introduces by a *wh*-word

Word level (part of speech)

CC	coordinating conjunction	CD	cardinal number
DT	determiner	IN	preposition
JJ	adjective	JJR	adjective, comparative
JJS	adjective, superlative	MD	modal
NN	noun, singular or mass	NNS	noun, plural
NNP	proper noun, singular	NNPS	proper noun, plural
PRP	personal pronoun	PRP$	possessive pronoun
RB	adverb	RBR	adverb, comparative
RBS	adverb, superlative	SYM	symbol
TO	*to*	VB	verb, infinitive
VBD	verb, past tense	VBG	verb, gerund
VBN	verb, past participle	VBZ	verb, present third person singular
WDT	*wh*-determiner	WP	*wh*-pronoun
WRB	*wh*-adverb		

APPENDIX B Dependency logic rules

Appendix B.1 presents the list of Prolog rules used in the comparison of relation extraction methods presented in Section 4.5. The dependency/grammatical roles used in the list are explained in Appendix B.2

B.1 Prolog typed dependency rules

```
% Rule 'X verb Y'
rel(X,Verb,Y) :- subject(Verb,X), object(Verb,Y),
  annotated(X), annotated(Y).
rel(X,Verb,Y) :- subject(Y,X), cop(Y,Verb),
  annotated(X), annotated(Y).

% Rule 'X not verb Y'
rel(X,Not,Verb,Y) :- subject(Verb,X), object(Verb,Y), neg(Verb,Not),
  annotated(X), annotated(Y), token(Not,'not').

% Rule 'X verb prep Y'
rel(X,Verb,Prep,Y) :- subject(Verb,X), prep(Verb,Prep), object(Prep,Y),
  annotated(X), annotated(Y).

% Rule 'X not verb prep Y'
rel(X,Not,Verb,Prep,Y) :- subject(Verb,X), prep(Verb,Prep), object(Prep,Y), neg(Verb,Not),
  token(Not,'not'), annotated(X), annotated(Y).

% Rule 'X verb to verb Y'
rel(X,Verb1,To,Verb2,Y) :- subject(Verb1,X), xcomp(Verb1,Verb2), aux(Verb2,To),
  object(Verb2,Y), token(To,'to'), annotated(X), annotated(Y).

% Rule 'X not verb to verb Y'
rel(X,Not,Verb1,To,Verb2,Y) :- subject(Verb1,X), xcomp(Verb1,Verb2), aux(Verb2,To),
  object(Verb2,Y), neg(Verb1,Not), token(To,'to'), token(Not,'not'),
  annotated(X), annotated(Y).

% Rule 'X verb to verb prep Y'
rel(X,Verb1,To,Verb2,Prep,Y) :- subject(Verb1,X), xcomp(Verb1,Verb2), aux(Verb2,To),
  prep(Verb2,Prep), object(Prep,Y), token(To,'to'), annotated(X), annotated(Y).

% Rule 'X not to verb prep Y'
rel(X,Not,Verb1,To,Verb2,Prep,Y) :- subject(Verb1,X), xcomp(Verb1,Verb2), aux(Verb2,To),
  prep(Verb2,Prep), object(Prep,Y), neg(Verb1,Not), token(To,'to'), token(Not,'not'),
  annotated(X), annotated(Y).

% auxiliary rules
subject(A,B) :- nsubj(A,B).
subject(A,B) :- nsubjpass(A,B).
subject(G,X) :- conj(Z,X), subject(G,Z).
subject(G,X) :- appos(Z,X), subject(G,Z).
subject(A,B) :- mod(B,A).
mod(A,B) :- partmod(A,B).
mod(A,B) :- amod(A,B).
object(A,B) :- dobj(A,B).
object(A,B) :- pobj(A,B).
object(G,Y) :- conj(Z,Y), object(G,Z).
object(G,Y) :- appos(Z,Y), object(G,Z).
```

B.2 Stanford grammatical role hierarchy

The following is a *selected* list of the Stanford grammatical role hierarchy [49]. The complete list with examples is provided in [50].

- dep - dependent
  - aux - auxiliary
    - cop - copula
  - conj - conjunct
  - cc - coordination
  - arg - argument
    - subj - subject
      - nsubj - nominal subject
        - nsubjpass - passive nominal subject
    - comp - complement
      - obj - object
        - dobj - direct object
        - pobj - object of preposition
      - xcomp - clausal complement with external subject
  - mod - modifier
    - amod - adjectival modifier
    - partmod - participial modifier
    - appos - appositional modifier
    - nn - noun compound modifier
    - advmod - adverbial modifier
      - neg - negation modifier
    - det - determiner
    - prep - prepositional modifier
  - sdep - semantic dependent
    - xsubj - controlling subject