Text Mining with Information Extraction

Un Yong Nahm

Abstract

The popularity of the Web and the large number of documents available in electronic form has motivated the search for hidden knowledge in text collections. Consequently, there is growing research interest in the general topic of text mining. In this paper, we develop a text-mining system by integrating methods from Information Extraction (IE) and Data Mining (Knowledge Discovery from Databases or KDD). By utilizing existing IE and KDD techniques, text-mining systems can be developed relatively rapidly and evaluated on existing text corpora for testing IE systems.

We present a general text-mining framework called DiscoTEX which employs an IE module for transforming natural-language documents into structured data and a KDD module for discovering prediction rules from the extracted data. When discovering patterns in extracted text, strict matching of strings is inadequate because textual database entries generally exhibit variations due to typographical errors, misspellings, abbreviations, and other sources. We introduce the notion of discovering "soft-matching" rules from text and present two new learning algorithms. TextRISE is an inductive method for learning soft-matching prediction rules that integrates rule-based and instance-based learning methods. Simple, interpretable rules are discovered using rule induction, while a nearest-neighbor algorithm provides soft matching. SoftApriori is a text-mining algorithm for discovering association rules from texts that uses a similarity measure to allow flexible matching to variable database items. We present experimental results on inducing prediction and association rules from natural-language texts demonstrating that TextRISE and SoftApriori learn more accurate rules than previous methods for these tasks. We also present an approach to using rules mined from extracted data to improve the accuracy of information extraction. Experimental results demonstate that such discovered patterns can be used to effectively improve the underlying IE method.

Contents

Chapter 1

Introduction

The recent abundance of digital information available electronically has made the organization of textual information into an important task. Text mining is a burgeoning new technology for discovering knowledge from text data. With the fast growth of the number of pages on the World Wide Web, text mining plays a key role in managing information and knowledge, and is therefore attracting increasing attention (Berry, 2003b, 2003a; Feldman, 1999; Hearst, 2003, 1999; Grobelnik, 2001, 2003; Mladenić, 2000; Muslea, 2004).

1.1 Text Data Mining and Information Extraction

Data Mining (DM) or Knowledge Discovery in Databases (KDD) is the process of identifying novel and understandable patterns in data (Han & Kamber, 2000; Witten & Frank, 1999). Data mining seeks not only information or answers to the question which the user already knows to ask, but discovers deep knowledge embedded within the data. In order to do that, data mining applies computational techniques, usually in the form of a learning algorithm, to find potentially useful patterns in the data. Most existing data mining approaches look for patterns in a relational table of data (Agrawal, Imielinsky, & Swami, 1993).

Text mining or text data mining, the process of finding useful or interesting patterns, models, directions, trends, or rules from unstructured text, is used to describe the application of data mining techniques to automated discovery of knowledge from text (Chakrabarti, 2002; Han & Kamber, 2000). Generally text mining has been viewed as a natural extension of data mining (Hearst, 2003, 1999). This reflects the fact that the advent of text mining relies on the burgeoning field of data mining to a great degree.

However, unlike data mining, which focuses on the well-structured collections that exist in either relational databases or data warehouses, text mining excavates data that is far less structured. Much of today's electronic data resides not in traditional relational databases, but "hidden" in the Web and natural-language documents. In this paper, we present a new framework for text mining based on the integration of traditional data mining and Information Extraction (IE).

The goal of an IE system is to find specific data in natural-language texts. The data to be extracted is typically given by a template which specifies a list of slots to be filled with substrings taken from the document. IE is useful for a variety of applications, particularly given the recent proliferation of Internet and web documents. Recent applications include course and research project homepages (Freitag, 1998a; Thompson, Smarr, Nguyen, & Manning, 2003), seminar announcements (Freitag, 1998b), apartment rental ads (Soderland, 1999), job announcements (Califf & Mooney, 1999), geographic web documents (Etzioni, Cafarella, Downey, Kok, Popescu, Shaked, Soderland, Weld, & Yates, 2004), government reports (Pinto, McCallum, Wei, & Croft, 2003), and medical abstracts (Bunescu, Ge, Kate, Marcotte, Mooney, Ramani, & Wong, 2004).

Traditional data mining assumes that the information to be "mined" is already in the form of a relational database. Unfortunately, for many applications, electronic information is only available in the form of unstructured natural-language documents rather than structured databases. IE addresses the problem of transforming a corpus of textual documents into a more structured database, thereby suggesting an obvious role that can be played in text mining when combined with standard KDD methods. In this paper, we suggest using an IE module to locate specific pieces of data in raw text, and to provide the resulting database to the KDD module for rule mining.

1.2 Heterogeneity of Text Data

In comparison with relational databases, natural-language corpora available on the internet are heterogeneous and noisy. Entries in many textual database fields could exhibit minor variations that can prevent mining algorithms from discovering important regularities. Variations can arise from typographical errors, misspellings, abbreviations, as well as from other sources.

Variations are particularly pronounced in data that is automatically extracted from unstructured or semi-structured documents or web pages (Ghani, Jones, Mladenić, Nigam, & Slattery, 2000; Nahm & Mooney, 2000). For example, in data on local job offerings that we automatically extracted from newsgroup postings, Windows operating system is variously referred to as "Microsoft Windows", "MS Windows", "Windows 95/98/ME", etc..

Some previous work has addressed the problem of identifying similar or duplicate records, where it is referred to as record linkage (Winkler, 1999), the merge/purge problem (Hernández & Stolfo, 1995), duplicate detection (Monge & Elkan, 1997), hardening soft databases (Cohen, Kautz, & McAllester, 2000), and reference matching (McCallum, Nigam, & Ungar, 2000b). Typically, a fixed textual similarity metric is used to determine whether two values or records are similar enough to be duplicates. In this approach, "Microsoft Windows", "MS Windows", and "Windows 95/98/ME" are mapped to a unique term as a pre-processing step.

We propose the alternative method of directly mining "dirty" data by discovering "soft-matching" rules whose antecedents and consequents are evaluated based on sufficient similarity to database entries. Similarity of text can be measured using standard "bag of words" metrics (Salton, 1989) or edit-distance measures (Gusfield, 1997); other standard similarity metrics can be used for numerical and additional data types. For instance, soft-matching rules such as "If *Windows* is in the list of required skills for a job, then knowledge for *IIS* is also required for that job." are discovered from a set of job announcements. In this case, "Windows" and "IIS" can be matched to similar strings such as "MS Windows" or "IIS Services" respectively.

1.3 Our Contribution

While there is a growing interest in the general topic of text mining, there are few working systems or detailed experimental evaluations. This paper introduces DiscoTEX, a new framework for text mining based on the integration of *Information Extraction* (IE) and traditional Knowledge Discovery from Databases (KDD), a.k.a. *data mining.* We explore the interaction between these two important techniques to perform text mining tasks by presenting an approach to using an automatically learned IE system to extract a structured databases from a text corpus, and then mine this database with traditional KDD tools. We show that text-mining systems can be developed relatively rapidly and evaluated on existing IE corpora, by utilizing existing IE and KDD technology.

To address the heterogeneity problem, we present a method, TextRISE, for learning soft-matching rules from text using a modification of the RISE algorithm (Domingos, 1996), a hybrid of rule-based and instance-based (nearest-neighbor) learning methods. Similarly, we introduce an algorithm, SoftApriori that discovers soft-matching *association rules* given a user-supplied similarity metric for each field. SoftApriori is a natural extension of the traditional association rule mining algorithm (Agrawal & Srikant, 1994) with soft-matching based on a specified similarity metric. With encouraging results from experiments in several domains, we show how these approaches can induce accurate predictive rules despite the heterogeneity of automatically extracted textual databases. By illustrating that soft-matching allows discovery of additional interesting rules, capturing certain relationships more accurately, we show that allowing the discovery of soft-matching rules can eliminate the need for certain types of tedious data cleaning prior to knowledge discovery.

We also explore a less obvious interaction between IE and KDD in the proposed text-mining framework. KDD can in turn provide benefits to IE as the predictive relationships between different slot fillers discovered by KDD provide additional clues about what information should be extracted from a document. This paper reports experiments in computer-related job and resumé domains demonstrating that predictive rules acquired by applying KDD to an extracted database can be used to improve the performance of the underlying information extraction system.

1.4 Organization

The rest of the paper is organized as follows. Chapter 2 will give a brief introduction to Text Mining and Information Extraction. An implementation of the DiscoTEX framework which simply combines IE and KDD will be described in Chapter 3 to demonstrate that the knowledge discovered from such an automatically extracted database is close in accuracy to the knowledge discovered from a manually constructed database. Chapter 4 presents and analyzes experimental results obtained with TextRISE, a more sophisticated implementation of the proposed text mining framework with partial matching rules incorporated in the prediction rule-learning algorithm. In Chapter 5, another approach called SoftApriori will be overviewed. SoftApriori discovers association rules from noisy textual databases. In Chapter 6, we will address the performance and the scalability issues which are important in real-world applications. Several optimization techniques employed in our system to speed up the running time will be discussed. After that, we will give experimental results obtained with our systems on internet documents such as Usenet newsgroup postings in Chapter 7. Chapter 8 presents initial results on the less obvious interaction between KDD and IE. We will show that rules mined by KDD can be used to improve the performance of the underlying IE. We will then review some related work or research problems briefly in Chapter 9 followed by a discussion on possible directions for future work in Chapter 10. Finally, we will discuss the significance of our research and make conclusions in Chapter 11.

Chapter 2

Background on Text Mining and Information Extraction

We will start by giving some basic concepts and overview for information extraction and existing text mining technology.

2.1 Information Extraction

The task of information extraction aims to find specific structured data in natural-language text. DARPA's Message Understanding Conferences (MUC) has concentrated on IE by evaluating the performance of participating IE systems based on blind test sets of text documents (DARPA, 1998). The data to be extracted is typically given by a template which specifies a list of slots to be filled with substrings taken from the document.

Usually the data to be extracted is described by a template specifying a list of slots to be filled, though sometimes it is specified by annotations in the document. In either case, slot-fillers may be of two types: they may be one of a set of specified values or they may be strings taken directly from the document.

Figure 2.1 shows a paired (shortened) document and template from an information extraction task in the job-posting domain. This template includes only slots that are filled by strings taken directly from the document. Several slots may have multiple fillers for the job-posting domain as in `(programming) languages`, `platforms`, `applications`, and `areas`.

IE has been shown to be useful in a variety of applications, e.g. seminar announcements, restaurant guides, course homepages, job postings, apartment rental ads, and news articles on corporate acquisition (Califf, 1999;

Document

```
Title: Web Development Engineer
Location: Austin, TX

This individual is responsible for design and implementation
of the web-interfacing components of the AccessBase server,
and general back-end development duties.

A successful candidate should have experience that includes:

   One or more of: Solaris, Linux, plus Windows/NT
   Programming in C/C++, Java
   Database access and integration: Oracle, ODBC
   CGI and scripting: one or more of Javascript,
                      Perl, PHP, ASP

Exposure to the following is a plus: JDBC, FrontPage and/or
Cold Fusion.

A BSCS and 2+ years experience (or equivalent) is required.
```

Filled Template

- title: "Web Development Engineer"
- location: "Austin, TX"
- languages: "C/C++", "Java", "Javascript", "Perl", "PHP", "ASP"
- platforms: "Solaris", "Linux", "Windows/NT"
- applications: "Oracle", "ODBC", "JDBC", "FrontPage", "Cold Fusion"
- areas: "Database", "CGI", "scripting"
- degree required: "BSCS"
- years of experience: "2+ years"

Figure 2.1: Sample text and filled template for a job posting

Ciravegna & Kushmerick, 2003; Kushmerick, 2001; University of Southern California, 1998). IE is also a suitable technology for automatically annotating web pages for the Semantic Web (Berners-Lee, Hendler, & Lassila, 2001; Stevenson & Ciravegna, 2003).

In particular, machine learning techniques have been suggested for extracting information from text documents in order to create easily searchable databases from the information, thus making the online text more accessible (Califf & Mooney, 1999). For instance, information extracted from job postings on the Web can be used to build a searchable database of jobs[1].

2.2 Learning for Information Extraction

Although most information extraction systems have been built entirely by hand until recently, automatic construction of complex IE systems has begun to be considered by many researchers lately (Califf, 1999; Ciravegna, Basili, & Gaizauskas, 2000; Kushmerick, 2001). By training on a corpus of documents annotated with their filled templates, they acquire a knowledge base of extraction rules that can be tested on novel documents.

Recent proliferation of research on information extraction implies the possibility of using a successfully-built IE component as part of a larger text-mining system. For instance, RAPIER (Califf, 1998) was were demonstrated to perform well on realistic applications such as USENET job postings and seminar announcements.

RAPIER (Robust Automated Production of Information Extraction Rules) (Califf, 1998) is a bottom-up relational rule learner for acquiring information extraction rules from a corpus of labeled training examples. It learns patterns describing constraints on slot fillers and their surrounding context using a specific-to-general search. Constraints on patterns can specify the specific words, part-of-speech, or semantic classes of tokens. The hypernym links in WordNet (Fellbaum, 1998) provide semantic class information and documents are annotated with part-of-speech information using the tagger of Brill (1994).

The learning algorithm of RAPIER was inspired by several inductive logic programming systems (Lavrac & Dzeroski, 1994). First, RAPIER creates most-specific patterns for each slot in each example specifying the complete word and tag information for the filler and its full context. New rules are created by generalizing pairs of existing rules using a beam search. When the best rule does not produce incorrect extractions, RAPIER adds it to the

[1]`http://flipdog.monster.com/`

rule base and removes existing rules that it subsumes. Rules are ordered by an information-theoretic heuristic weighted by the rule size. By training on a corpus of documents annotated with their filled templates, Rapier acquires a knowledge base of extraction rules that can then be tested on novel documents.

2.3 Rule Mining

Among various data mining techniques, we focus on the problem of finding useful rules from given data sets because rules are more comprehensible to human users. For a data set describing customer behavior in a supermarket, a rule or pattern might be, "10 percent of the customers buy diapers and beer together," and for the Web log data set, a rule could be, "If a person visits the UTCS web site, there is a 30% chance the person will visit the UTCS AI lab web site in the same week." There has been a large volume of research on association rule mining (Agrawal et al., 1993) and predictive rule induction (Mitchell, 1997).

2.3.1 Inductive Rule Learning

IF-THEN rules are one of the most expressive representations for learned hypotheses (Mitchell, 1997). For example, given a database of customer credit information, classification rules can be learned to label customers having low credit ratings. The rules can be used to categorize previously unseen data, e.g. future customers. In general, classification rule learning methods either extract rules from decision trees, adopt sequential set covering algorithms, or translate neural nets into human-readable rules. Most of them assume that examples are represented as "feature vectors", the components of which are either real numbers or nominal values.

Two widely-used schemes for rule-learning are C4.5rules (Quinlan, 1993) and Ripper (Cohen, 1995). Both of them generate concise and human-readable outputs, rule sets. C4.5rules induces rules from binary data by learning decision trees and translating them into pruned rules. The algorithm generates a set of rules for each path from a tree learned by C4.5. It then checks if the rules can be generalized by dropping conditions. C4.5 can handle training data with continuous attributes or missing attribute values and has been successfully applied to a wide variety of machine-learning tasks (Mitchell, 1997).

Ripper is a fast rule learner with an ability to handle set-valued features (Cohen, 1996b). Ripper, based on the incremental reduced error pruning algorithm, splits the available training data into a growing set and pruning set before learning rules. It is shown that RIPPER scales nearly linearly with the number of examples in the training set. Particularly on noisy data sets, RIPPER was shown to be equally accurate but more efficient than C4.5RULES (Cohen, 1995).

2.3.2 Association Rule Mining

Association rule mining is one of the most popular techniques in data mining (Han & Kamber, 2000; Witten & Frank, 1999). The problem of mining association rules is to discover all association rules that have support and confidence greater than the user-specified minimum support and minimum confidence.

An association rule is intended to capture dependence among items in a database. Specifically, we say that $i_1 \Rightarrow i_2$ if 1) i_1 and i_2 occur together in at least $s\%$ of the n baskets where baskets of items are subsets of the set of all items and 2) of all the bakskets containing i_1, at least $c\%$ also contains i_2. We call the probability that a basket contains both items (s) the *support* and the probability that a basket containing one item also contains the other (c) the *confidence*. The associations between items can be easily generalized to those among item sets.

The classical application of association rule mining techniques is market basket analysis about finding associations between items purchased by customers. Each basket in the previous definition may be viewed as a transaction that occurs in the supermarket. An association rule from a supermarket database, "$beer \Rightarrow pretzels$ $[20\%, 80\%]$" indicates that 20% (support) of customers bought beer and pretzels together and 80% (confidence) of those who bought beer also bought pretzels.

One of the popular algorithms for discovering association rules is APRIORI (Agrawal & Srikant, 1994) where the downward closure property was utilized to prune unnecessary branches for further consideration. APRIORI is based on breadth-first search and therefore ensures that the support values of all subsets of a candidate are known in advance. At each stage k, all candidates of a cardinality k are counted in a scan over the database. APRIORI prunes all candidate itemsets such that any subset of that itemset is not frequent.

2.4 Rule Mining from Text

Much text mining or knowledge discovery in text paradigms have been based on simple forms of text categorization as in KDT (Feldman & Dagan, 1995). However, recently several researchers have applied traditional rule induction methods to discover relationships from textual data. FACT (Feldman & Hirsh, 1996) discovers rules from text using *association rule mining.* For example, it discovered rules such as "Iraq $\Rightarrow$ Iran", and "Kuwait and Bahrain $\Rightarrow$ Saudi Arabia" from a corpus of Reuters news articles.

Ahonen, Heinonen, Klemettinen, and Verkamo (1998) also applied existing data mining techniques to discover *episode rules* (Mannila, Toivonen, & Verkamo, 1997) from text. For example: "chemicals, processing $\Rightarrow$ storage [2 - 3]" (If "chemicals" and "processing" occurs within two words, the word "storage" co-occurs within three words.) is an episode rule discovered from a collection of Finnish legal documents. Episode rule mining is used for language analysis because it preserves the sequential structure of terms in a text document.

In addition, decision tree methods such as C4.5 and C5.0, and rule learners such as FOIL and RIPPER have been used to discover patterns from textual data (Nahm & Mooney, 2000; Ghani et al., 2000). Using C5.0, Ghani et al. (2000) discovered interesting patterns, e.g. "Aerospace/defense companies are located in Florida", from the `Hoovers.com` online resource about companies. A first-order rule learner, FOIL was used to learn function-free Horn clauses (Quinlan & Cameron-Jones, 1993).

The relevant application areas of text mining include biomedical applications (Hahn, Romacker, & Schulz, 2002; Leroy, Chen, & Martinez, 2003; Muresan & Klavans, 2002; Scheffer & Leser, 2003; Schwartz & Hearst, 2003), web mining and personalization (Chakrabarti, 2002; Chiang, Laender, & Lim, 2003; Eirinaki & Vazirgiannis, 2003; Grobelnik, 2003), tools for natural language processing e.g. question-answering systems (Harabagiu, Bunescu, & Maiorano, 2001; Lin & Pantel, 2001), and business applications (Sullivan, 2000) such as customer relationship management and opinion mining (Dave, Lawrence, & Pennock, 2003).

2.5 Similarity Metrics

Most of the existing text mining techniques discover rules requiring an exact match. However, due to the heterogeneity problem disussed in Section 1.2, a form of soft-matching is needed to construct an effective text mining system.

Soft-matching requires a method to determine the "distance" between two textual items or documents.

Similarity of text can be measured using standard "bag of words" (BOW) metrics (Salton, 1989) or edit-distance measures (Sankoff & Kruskal, 1983) such as character edit distance used in the CORA research paper search engine (McCallum, Nigam, Rennie, & Seymore, 2000a). This section gives a brief overview of these standard text-similarity metrics.

2.5.1 Edit Distance

Edit distance is a well-known measure of the similarity of strings. It is based on elementary edit operations such as insertions, deletions, and substitutions where costs are associated with these edit operations. The distance between two strings is defined as the transformation from one string to another using edit operations with minimal costs. The greater the distance, the more different the strings are.

However, edit-distance cannot be used directly since it returns 0 if the strings are identical and greater values when they are different. Therefore, we define $similarity(x, y)$ as follows:

$$similarity(x, y) = 1 - normalized\_edit\_distance(x, y) \tag{2.1}$$

where normalized edit-distance is scaled to always be between 0 and 1 based on the lengths of the two strings:

$$normalized\_edit\_distance(x, y) = \frac{edit\_distance(x, y)}{maximum\_distance(x, y)} \tag{2.2}$$

The notion of maximum edit distance (as a dissimilarity measure) is introduced. Maximum distance between two strings ($maximum\_distance(x, y)$) is defined as the maximum possible value for edit distance between two strings x' and y', where $|x| = |x'|$ and $|y| = |y'|$.

Levenshtein distance (Levenshtein, 1966) is one of the well-known edit-distance functions. Levenshtein distance is defined to be the number of character deletions, insertions, or substitutions required to transform a string s_1 into another, s_2. For example, if s_1 is "windows" and s_2 is "windows", then Levenshtein distance between s_1 and s_2 is 0, because no transformations are needed. If s_1 is "windows" and s_2 is "windowsnt", then the distance is 2, because two insertions ("n" and "t") are sufficient to transform s_1 into

s_2. The Levenshtein edit-distance algorithm has been used in several text-processing tasks such as spell checking (Schulz & Mihov, 2002) and speech recognition (Robertson, Wong, Chung, & Kim, 1998).

Among various edit-distance functions, we use affine gap cost (Monge & Elkan, 1996; Needleman & Wunsch, 1970), an edit distance originally developed for gene/protein sequence comparison. Affine gap cost incurs one penalty for starting a new gap (i.e. sequence of deletions) and a typically smaller penalty for continuing an existing gap (i.e. contiguous deletions). Different edit operations have varying significance in different domains (Bilenko & Mooney, 2003). It is known that affine gap cost provides more intuitive results than other standard edit-distances, such as Levenshtein distance, for text strings (Nahm, Bilenko, & Mooney, 2002).

The computation of affine gap cost is performed by dynamic programming in time $O(nm)$ when n and m are the lengths of the two input strings as shown in Figure 2.2 (Needleman & Wunsch, 1970).

2.5.2 Vector Space

The vector-space model is typically used in Information Retrieval (IR) (Salton, 1989) to determine the similarity of two documents. In this model, a text is represented as a vector of real numbers, where each component corresponds to a word that appears in the set of all documents and the value is its frequency in the document. This is also known as a *bag-of-words* representation. The similarity of two documents x and y is the *cosine of the angle* between two vectors $\vec{x}$ and $\vec{y}$ representing x and y respectively, and calculated by the following formula:

$$similarity(x, y) = \frac{\vec{x} \cdot \vec{y}}{\mid \vec{x} \mid \times \mid \vec{y} \mid} \tag{2.3}$$

where $\mid \vec{x} \mid$ and $\mid \vec{y} \mid$ are the norms of each document vector. Cosine distance is defined as one minus the cosine of the included angle between vectors.

$$cosine\_distance(x, y) = 1 - cosine\_similarity(x, y) \tag{2.4}$$

The TFIDF (Term Frequency, Inverse Document Frequency) weighting scheme (Salton, 1989) is used to assign higher weights to distinguished terms in a document. TFIDF makes two assumptions about the importance of a term. First, the more a term appears in the document, the more important

Input: s_1 and s_2 are the given strings.
Output: *cost* is the affine gap cost between s_1 and s_2.
Parameters: *match_cost*, *mistmatch_cost*, *gap_start_cost*, *gap_extend_cost*.

```
Function AffineGapCost ($s_1$, $s_2$)

$m := length(s_1).$; $n := length(s_2).$
If ($m = 0$) or ($n = 0$)
   $cost := gap\_start\_cost + (m + n - 1) \times gap\_extend\_cost.$
   Return $cost.$
For ($j := 0$; $j < n + 1$; $j{+}{+}$) do
      $I[0][j] := MAX.$; $D[0][j] := MAX.$
For ($j := 0$; $j < m + 1$; $j{+}{+}$) do
      $I[j][0] := MAX.$; $D[j][0] := MAX.$
$T[0][0] := 0.$; $T[0][1] := gap\_start\_cost.$; $T[1][0] := gap\_start\_cost.$
For ($j = 2$; $j < n + 1$; $j{+}{+}$) do
      $T[0][j] := T[0][j-1] + gap\_extend\_cost.$
For ($j := 2$; $j < m + 1$; $j{+}{+}$) do
      $T[j][0] := T[j-1][0] + gap\_extend\_cost.$
For ($i := 1$; $i < m + 1$; $i{+}{+}$) do
      For ($j = 1$; $j < n + 1$; $j{+}{+}$) do
            If ($D[i-1][j] + gap\_extend\_cost > T[i-1][j] + gap\_start\_cost$)
               $D[i][j] := T[i-1][j] + gap\_start\_cost.$
            Else
               $D[i][j] := D[i-1][j] + gap\_extend\_cost.$
            If ($I[i][j-1] + gap\_extend\_cost > T[i][j-1] + gap\_start\_cost$)
               $I[i][j] := T[i][j-1] + gap\_start\_cost.$
            Else
               $I[i][j] : I[i][j-1] + gap\_extend\_cost.$
            If ($s1[i-1] = s2[j-1]$)
               $sub\_cost := match\_cost.$
            Else
               $sub\_cost := mismatch\_cost.$
            If ($T[i-1][j-1] + sub\_cost < D[i][j]$) and ($T[i-1][j-1] + sub\_cost < I[i][j]$)
               $T[i][j] := T[i-1][j-1] + sub\_cost.$
            Else If $D[i][j] < I[i][j]$ Then $T[i][j] := D[i][j].$
                  Else                       $T[i][j] := I[i][j].$
$cost := T[m][n].$
Return $cost.$
```

Figure 2.2: The algorithm for computing affine gap cost

it is (*term frequency*). Second, the more it appears through the entire collection of documents, the less important it is since it does not characterize the particular document well (*inverse document frequency*). In the TFIDF framework, the weight for term t_j in a document d_i, w_{ij} is defined as follows:

$$w_{ij} = tf_{ij} \times \log_2 \frac{N}{n} \tag{2.5}$$

where tf_{ij} is the frequency of term t_j in document d_i, N is the total number of documents in a collection, and n is the number of documents in which term t_j occurs at least once.

Chapter 3

DiscoTEX: Combining IE and KDD for Text Mining

In this chapter, we suggest a new framework for text mining based on the integration of Information Extraction (IE) and traditional Knowledge Discovery from Databases (KDD). We first present the idea of combining IE and KDD serially for text mining, explain how a document in this sytem can be represented as a vector of textual elements, and empirically show that rules mined from IE-extracted data are nearly as accurate as those discovered from manually extracted data.

3.1 Introduction

As previously stated in Chapter 1, the assumption of traditional data mining that the information to be mined is already in the form of a relational database does not hold in many cases. For a number of applications, electronic information is available only in the form of unstructured natural-language documents which cannot be directly analyzed by statistical data mining methods. Information Extraction, a task that has attracted increasing attention since the start of the Message Understanding Conferences (MUCs) (DARPA, 1998), addresses the problem of transforming a corpus of textual documents into a more structured database.

Since structured databases transformed from unstructured texts by information extraction can be supplied to traditional data mining as input, IE can play an essential role in data preparation for text mining as illustrated in Figure 3.1. In the proposed IE-based text-mining framework, called DiscoTEX (Discovery from Text EXtraction), the IE module identifies specific

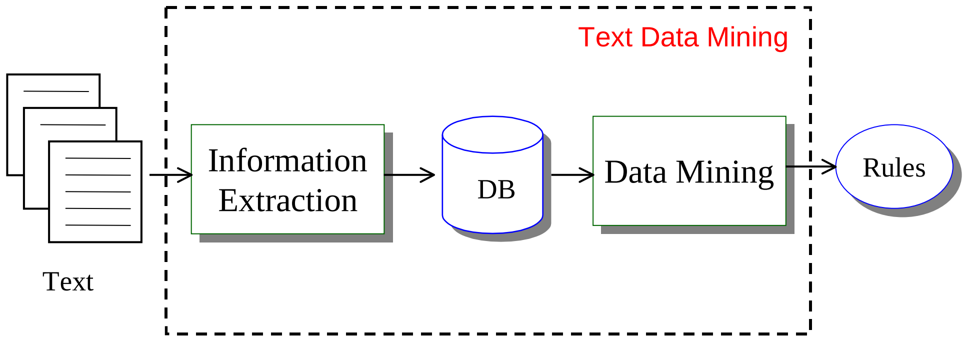


Figure 3.1: Overview of IE-based text mining framework

pieces of data in raw text, and the resulting database is provided to the KDD module for further mining of knowledge. Although constructing an IE system is a difficult task, there has been significant recent progress in using machine-learning methods to help automate the construction of IE systems as shown in Section 2.2. By manually annotating a small number of documents with the information to be extracted, a fairly accurate IE system can be induced from this labeled corpus and then applied to a large body of raw text to construct a large database for mining. In this way, a small amount of labeled training data for an IE learning system can be automatically transformed into a large database of structured information ready to be mined with traditional KDD methods.

General IE learning systems such as Rapier (Califf & Mooney, 1999) or BWI (Freitag & Kushmerick, 2000) can be used to construct an IE module for DiscoTEX. After constructing an IE system that extracts the desired set of slots for a given application, a database is constructed from a corpus of texts by applying the extractor to each document to create a collection of structured records. Standard KDD techniques such as C4.5rules (Quinlan, 1993) or Ripper (Cohen, 1995) can then be applied to the resulting database to discover interesting relationships.

Book Description from `Amazon.com`

Title: Harry Potter and the Order of the Phoenix (Book 5)
Author: J. K. Rowling, Mary GrandPre
Comments: This book was the best book I have ever read.
If you are in for excitement this book is the one you want to read.
Synopsis: As his fifth year at Hogwarts School of Witchcraft and Wizardry approaches, 15-year-old Harry Potter is in full-blown adolescence. Harry is feeling especially edgy at the lack of news from the magic world, wondering when the freshly revived evil Lord Voldemort will strike. Returning to Hogwarts will be a relief... or will it?
Subject: Fiction, Mystery, Magic, Children, School,
Juvenile Fiction, Fantasy, Wizards
Publication Year: 2003

Figure 3.2: An example of a book description

3.2 Data Representation

3.2.1 Representation

An interesting question is how to represent a document or a textual data in text-mining systems. Most existing IE learning systems represent a document as a sequence of characters or tokens (Califf & Mooney, 1999; Freitag & Kushmerick, 2000; Muslea, 1999). Since the DiscoTEX framework relies on an IE system as a preprocessing module, a natural way to handle data is to treat the slot-values as sequences of characters, i.e. strings. However, for many applications, much larger strings are often identified as shown in Figure 3.2 [1]. In this example, slots such as `comments` or `synopsis` contain long strings which are difficult to deal with when they are simply treated as sequences of characters.

A classical way of handling long strings is to treat them as "bag-of-words" (Salton, 1989). Standard approaches to text categorization and information retrieval makes use of the bag-of-words (BOW) text representation technique that maps a document to a high dimensional feature vector, where each entry of the vector represents the frequency of a term. This approach only retains the frequency of the terms in the document while losing the information on the order of the terms. The BOW model is usually accompanied by the removal of non-informative words (stop-words) and by

[1] `http://www.amazon.com/`

the optional replacing of words by their stems. On the other hand, many wrapper-learning systems represent a document as a linear sequence of tokens (Cohen, Hurst, & Jensen, 2002; Muslea, Minton, & Knoblock, 1999) as they are more concerned with the structural cues based on special characters such as carriage returns.

One problem is that different applications need different representations. To allow more flexibility in our text mining framework, we augment the feature vector model of traditional machine learning approaches (Mitchell, 1997; Witten & Frank, 1999) with the bag-of-words model which is the most common scheme for representing long documents, the token-based model for preserving the order of terms, and the simple sequence-of-characters model for shorter strings. Users are able to specify which model should be applied to each slot in advance. For example, we can use string edit-distance as the similarity metric for shorter strings and cosine similarity for longer fields. One advantage of this approach is that a new type of document representation and its similarity metric can be easily plugged into the system.

Specifically, we represent an IE-processed document as a vector of slot-values, one for each slot filler. A rule can be represented as an antecedent that is a conjunction of slot-values for some subset of slots and a conclusion that is a predicted slot-value for another slot. Sometimes multiple fillers can be identified for a slot in many domains. In that case, a slot-value corresponds to a set of textual items. For example, the author slot of the example shown in Figure 3.2 could have two fillers or items, "J. K. Rowling" and "Mary Grandpre" rather than treating them as one big bag of words or a long string. To allow multiple items for each slot, we extend the simple "vector-of-slot-values" model so that a slot-value can contain a set of distinct items.

To summarize, we model documents as vectors of slot-values where each slot-value corresponds to each slot of the information extraction system as shown in Figure 3.3 with the Backus Naur Form (BNF) notation. Each slot can be either an item or a set of items which can be either long documents, short strings, or numbers. In our system, we modeled each filler as either 1) long documents which are represented using the vector-space model (BOW Model), 2) a list of tokens (Token Model), 3) short strings as a list of characters (String Model), or 4) numbers including dates (Numerical Model).

Representation

```
<document> :: <document> <slot-value> | empty .
<slot-value> ::  <item> | <slot-value> <item> .
<item> :: <bow> | <string> | <token> | <number> .
```

Figure 3.3: Document representation

Model	Representation	Similarity Metric
BOW	Bag-of-Words	Cosine Similarity
String	Sequences of Characters	Character-level Edit-Distance
Token	Sequences of Tokens	Token-level Edit-Distance
Number	Numbers	Numeric Distance

Table 3.1: Document models and corresponding similarity metrics

3.2.2 Data Types

Table 3.1 summarizes the models and the corresponding similarity metrics for textual items.

BOW Model

The BOW model follows the vector space model of handling long strings as "bags-of-words" (Salton, 1989). In the BOW model, we eliminate 524 commonly-occurring stop-words (e.g. "the", "is", and "you") but do not perform stemming. Standard set-operations are extended to bags in the obvious way (Peterson, 1976). For example, the intersection of two bags is defined as a bag that contains as many as the minimum of elements in both bags. The similarity between two slot-values with the BOW model is measured by computing the cosine similarity of two BOWs.

String Model

The string model represents short strings as a list of characters. The similarity metric for the string model is the character-level edit distance.

Token Model

The token model used in some wrapper learning system is also introduced. In our model, a token list is defined as an ordered sequence of tokens x_1, x_2,

... x_n, where x_i in T (set of tokens) is a term. Similarly, a string is defined as an ordered sequence of characters y_1, y_2, ..., y_n, where y_j is a character. Note that the token space for the token model shares the same set of terms in the BOW model. The edit-distance measure described in Section 2.5.1 can be applied to both sequences of tokens and characters for computing simliarities between two items.

Number Model

The number model represents numerical values. The numerical difference is used to measure the similarity between two numbers.

3.3 Data Sets

In this section, we present the four data sets, job-postings, resumés, book descriptions, and movie descriptions used in the experiments that will be presented later.

3.3.1 Job-postings Data Set

600 computer-science job postings to the newsgroup `austin.jobs` originally collected and manually annotated for training Rapier (Califf, 1998) were used. Information on programming languages, platforms, applications, areas, company, recruiter, job title, required years of experience, desired years of experience, salary, post date, city, state and country were identified to construct a textual database of job requirements.

Since `austin.jobs` is not a moderated newsgroup, not all posted documents are relevant to our task. Some of them are resumés posted by job-seekers, advertisements, or non-computer-science job postings. Therefore, before constructing a database using an IE system, we filtered out irrelevant documents from the newsgroup using a trained text categorizer. First, 1,000 postings were collected and classified by a human expert as relevant or irrelevant. Next, a bag-of-words Naive-Bayes text categorizer (Mitchell, 1997; McCallum & Nigam, 1998) was trained on this data to identify relevant documents (spam postings, resumés, or non-cs job postings). The resulting categorizer has an accuracy of over 99% and is used to filter irrelevant documents from the original postings.

Slot	Model
Job title	String
Programming languages	String (Multiple)
Platforms	Strings (Multiple)
Applications	Strings (Multiple)
Areas	String (Multiple)
Company	String
Recruiter	String
Required years of experience	Number
Desired years of experience	Number
Salary	Number
Post date	String
City	String
State	String
Country	String

Table 3.2: Slots and slot-value types for job-postings data set

3.3.2 Resumé Data Set

300 user-annotated computer-science resumé postings to the newsgroup `misc.job.resumes`, `alt.resumes`, and `us.job.resumes` were collected. We used a simple web-crawler for spidering `groups.google.com` web site in order to collect documents from the newsgroups. A bag-of-words Naive-Bayes text categorizer (McCallum & Nigam, 1998) is used again to identity relevant documents. Similar information to that of the job-postings data set are extracted as shown in Table 3.3.

Slot	Model
Name	String
Programming languages	String (Multiple)
Platforms	Strings (Multiple)
Applications	Strings (Multiple)
Areas	String (Multiple)
Hardware	String (Multiple)
Company	String
Recruiter	String
City	String
State	String

Table 3.3: Slots and slot-value types for resumé data set

Slot	Model
Title	String
Author	Token (Multiple)
Type	Strings
Publisher	String
Publication date	String
Subjects	String (Multiple)
Related books	String (Multiple)
Related authors	String (Multiple)
Price	Number
Average rating	Number
Reviews	BOW (Multiple)
Synopsis	BOW (Multiple)
Comments	BOW (Multiple)

Table 3.4: Slots and slot-value types for book-descriptions data set

3.3.3 Book Data Set

12,000 book descriptions automatically extracted from the `Amazon.com` online bookstore for a book recommending system (Mooney & Roy, 2000) are used. The information extractor (wrapper) for Amazon was developed manually and is highly accurate. 10 fields (title, author, type, publisher, publication date, subjects, related books, related authors, price, and average rating, reviews, synopsis, comments) are identified as shown in Table 3.4.

3.3.4 Movie Data Set

The movie data set is drawn from the Internet Movie Database (`IMDb.com`). 7,000 movie descriptions with plot summaries are automatically extracted. 7 fields (title, director, writer, genres, keyword, plot, year) are identified as shown in Table 3.4.

3.4 Initial DiscoTEX

System Architecture

In the experiments in this section, Rapier (Califf & Mooney, 1999) is used to construct an IE module for DiscoTEX. Rapier was trained on only 60 labeled documents, at which point its accuracy at extracting information is somewhat limited; extraction precision (percentage of extracted slot fillers that are correct) is about 91.9% and extraction recall (percentage of all of the

Slot	Model
Title	String
Director	String (Multiple)
Writer	Strings (Multiple)
Genres	String (Multiple)
Keyword	String (Multiple)
Plot	BOW
Year	Number

Table 3.5: Slots and slot-value types for movie-descriptions data set

correct fillers extracted) is about 52.4% . We purposely trained Rapier on a relatively small corpus in order to demonstrate that labeling only a relatively small number of documents can result in a learned extractor capable of building a database from which accurate knowledge can be discovered.

In order to discover prediction rules, we treat each slot-value pair in the extracted database as a distinct binary feature. For instance, given a set of n job postings, we could go through every posting and list the job skills that it has and does not have. We can represent a single postings's list of required job skills by a simple binary vector which has a 1 in the ith slot if the postings has the ith skill specified, and a 0 otherwise. In this way, the n job-posting messages are converted into n different binary vectors. After a set of binary vectors are obtained through the conversion, rules are learned for predicting each feature of the vectors from all other features.

Similar slot fillers are first collapsed into a pre-determined standard term. For example, "Windows 98" is a popular filler for the `platform` slot, but it often appears as "MS Win 98", "Win 98", "Win98", and so on, and "DBA" in the `title` slot is an abbreviation for "DataBase Administrator". These terms are collapsed to unique slot values before prediction rules are mined from the data. A small domain-dependent synonym dictionary is used to identify such similar terms. Trivial cases such as "Databases" → "Database" and "Client/Server" → "Client-Server" are handled by manually contrived synonym-checking rules.

We have applied C4.5rules (Quinlan, 1993) and Ripper (Cohen, 1995) to induce rules from the resulting binary data. Ripper runs significantly faster since it has an ability to handle *set-valued features* (Cohen, 1996b) to avoid the step of explicitly translating slot fillers into a large number of binary features. Specifically, rules are induced for predicting each piece of information in each database field given all other information in a record. In general, any standard classification rule-learning methods can be employed

- Oracle $\in$ application **and** QA Partner $\in$ application $\rightarrow$ SQL $\in$ language
- $C++$ $\in$ language **and** C$\in$language **and** CORBA $\in$ application $\rightarrow$ Windows inplatform
- HTML $\in$ language **and** WindowsNT $\in$ platform **and** Active Server Pages $\in$ application $\rightarrow$ Database$\in$area
- UNIX $\notin$ platform **and** Windows $\notin$ platform **and** Games $\in$ area $\rightarrow$ 3D$\in$area
- Java $\in$ language **and** ActiveX $\in$ area **and** Graphics $\in$ area $\rightarrow$ Web $\in$ area

Figure 3.4: Sample rules mined for computer-science job postings

for this task.

Sample Rules

Discovered knowledge describing the relationships between slot values is written in a form of production rules. If there is a tendency for `Web Design` to appear in the area slot when `Director` appears the in applications slot, this is represented by the production rule, `Director`$\in$`application` $\rightarrow$ `Web Design`$\in$`area`. Rules can also predict the absence of a filler in a slot. Sample rules mined by C4.5RULES from a database of 600 jobs extracted from the USENET newsgroup `austin.jobs` are shown in Figure 3.4.

In the initial DiscoTEX, documents annotated by the user are provided to Rapier as training data. IE rules induced from this training set are stored in the IE rule base and subsequently used by the extraction module. The learned IE system then takes unlabeled texts and transforms them into a database of slot-values, which is provided to the KDD component (i.e. C4.5 or Ripper) as a training set for constructing a knowledge base of prediction rules. The training data for KDD can include the user-labeled documents used for training IE, as well as a larger IE-labeled set automatically extracted from raw text.

3.5 Automatically Extracted Data vs. Manually Extracted Data

The accuracy of current IE systems, whether built manually or induced from data, is limited. Therefore, an automatically extracted database will inevitably contain significant numbers of errors. An important question is whether the knowledge discovered from this “noisy” database is significantly less reliable than knowledge discovered from a cleaner traditional database.

Slots	AvgNumFiller	AvgNumDoc	NumFiller
language	0.13	2.30	80
platform	0.17	7.11	104
application	0.30	3.76	179
area	0.60	1.17	361
total	1.21	1.38	724

Table 3.6: Statistics on slot-fillers

In this section, we present experiments on the job postings domain (Section 3.3.1) demonstrating that knowledge discovered from an automatically extracted database is close in accuracy to that discovered from a manually constructed database with a simple implementation of the DiscoTEX framework. Since all the extracted items in this domain are short strings, they are represented as simple strings (sequences of characters).

3.5.1 Experimental Methodology

Discovered knowledge is only useful and informative if it is accurate. Discovering fluke correlations in data is not productive, and therefore it is important to measure the accuracy of discovered knowledge on independent test data. The primary question we address in the experiments in this section is whether knowledge discovered from automatically extracted data (which may be quite noisy) is relatively reliable compared to knowledge discovered from a manually constructed database.

Ten-fold cross validation was used to generate training and test sets for extraction from the set of documents. Rules were mined for predicting the fillers of the `languages`, `platforms`, `applications`, and `areas` slots, since these are usually filled with multiple items that have potential predictive relationships. The total number of slot-values used in the experiment is 476: 48 slot-values are for languages slot, 59 for platforms, 159 for applications, and 210 for areas. Statistics on these slot-fillers are shown in Table 3.6, including the average number of fillers per document, average number of documents per filler, and the total number of distinct filler strings in the corpus.

In order to test the accuracy of the discovered rules, they are used to predict the information in a disjoint database of user-labeled examples. For each test job, each possible slot-value is predicted to be present or absent given information on all of its other slot-values. Average performance across

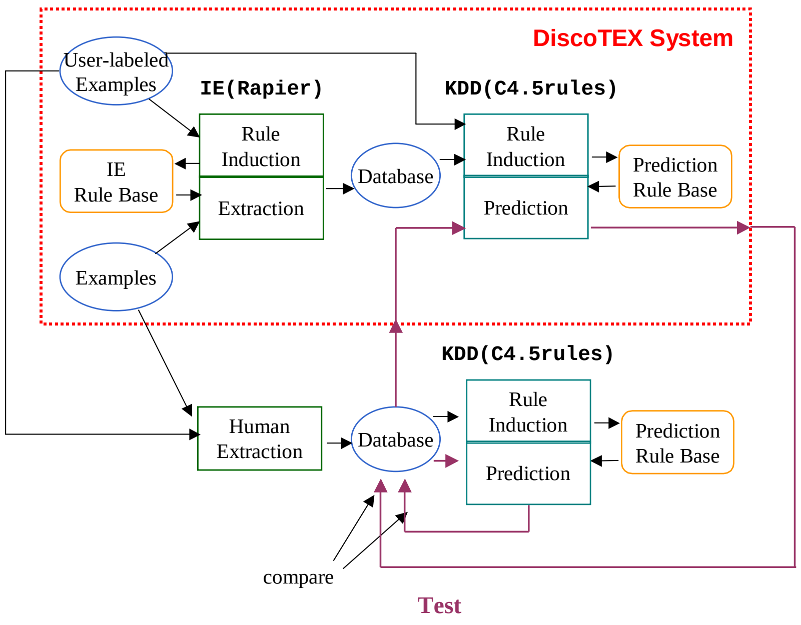


Figure 3.5: The system architecture for evaluation

all features and all test examples is then computed. The rules produced by RIPPER and C4.5RULES were found to be of similar accuracy, and the experiments in this section employ RIPPER since its computational time and space complexity is significantly less. The overall architecture of the system for evaluation is shown in Figure 3.5.

The classification accuracy for predicting absence or presence of slot fillers is not a particularly informative performance metric since high accuracy can be achieved by simply assuming every slot filler is absent. For instance, with 60 user-labeled examples, DISCOTEX gives a classification accuracy of 92.7% while the all-absent strategy has an accuracy of 92.5%. This is because the set of potential slot fillers is very large and not fixed in advance, and only a small fraction of possible fillers is present in any given example. Therefore, we evaluate the performance of DISCOTEX using the IE performance metrics of precision, recall, and F-measure with regard to predicting slot fillers. These metrics are defined as follows:

$$precision = \frac{number\_of\_present\_slot\_values\_correctly\_predicted}{number\_of\_slot\_values\_predicted\_to\_be\_present} \quad (3.1)$$

	Present	Absent
Predicted To Be Present	$m \times p$	$(n-m) \times p$
Predicted To Be Absent	$m \times (1-p)$	$(n-m) \times (1-p)$

Table 3.7: The expected outcome for random guessing

$$recall = \frac{number\_of\_present\_slot\_values\_correctly\_predicted}{number\_of\_present\_slot\_values} \tag{3.2}$$

F-measure is the harmonic mean of precision and recall and is computed as follows (when the same weight is given to precision and recall):

$$F\!-\!measure = \frac{2 \times precision \times recall}{precision + recall} \tag{3.3}$$

In order to obtain non-trivial bounds on precision and recall, a simple random guessing method is used as a benchmark. This approach guesses a slot-value based on its frequency of occurrence in the training data. For instance, if "Java" occurs as a programming language in 29% of jobs in the training data, then this approach guesses that it occurs 29% of the time for the test data. Instead of simulating this method, we analytically calculated its expected precision and recall for each slot-value. The expected outcome for this strategy for a given slot-value is summarized in Table 3.7, where p is the percentage of times the slot-value appears in the training examples, n is the total number of the test examples and m is the number of times the slot-value occurs in the test data.

Using the information in the table, the precision and the recall for random-guessing is determined as follows:

$$precision = \frac{m \times p}{(m \times p) + ((n-m) \times p)} = m/n \tag{3.4}$$

$$recall = \frac{m \times p}{(m \times p) + (m \times (1-p))} = p \tag{3.5}$$

Therefore, the benchmark precision for a slot-value is its probability of occurrence as estimated from the test data and the recall is its probability of occurrence as estimated from the training data. The only difference between the two is due to sampling error.

3.5.2 Results and Discussion

Because of the two different training phases used in DiscoTEX, there is a question of whether or not the training set for IE should also be used to train

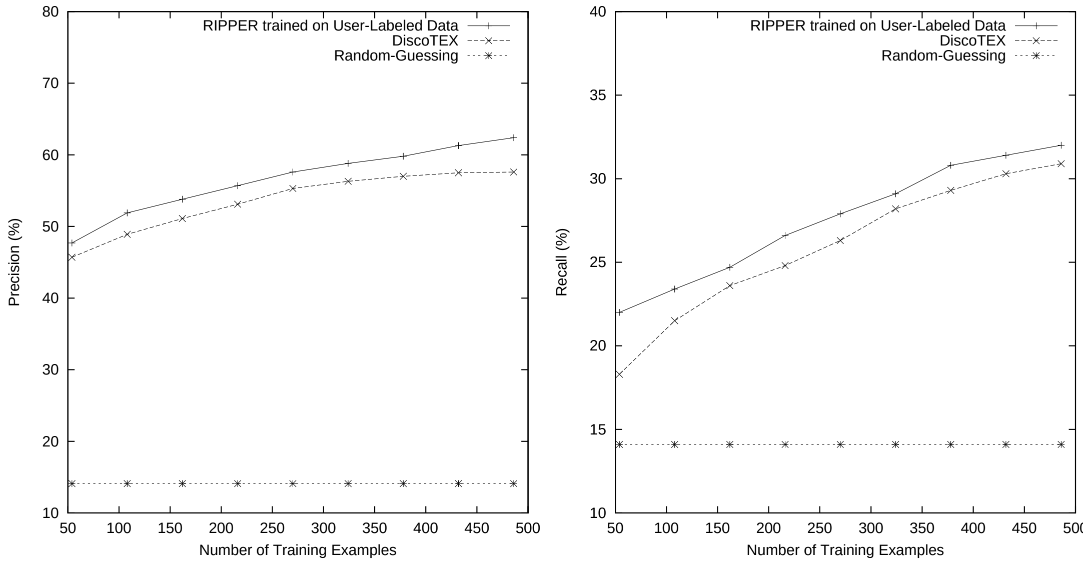


Figure 3.6: Precision and recall with disjoint IE training set

the rule-miner. In realistic situations, there is no reason not to use the IE training data for mining since the human effort has already been expended to correctly extract the data in this text. However, to clearly illustrate the difference between mining human-labeled and IE-labeled data, we first show a comparison with a disjoint IE training set. In this experiment, the IE training data are thrown away once they have been used to train RAPIER, since the extractor is unlikely to make the normal number of extraction errors on this data. Ten-fold cross-validation is performed on the remaining 540 examples in order to evaluate data mining. In order to clearly illustrate the impact of mining automatically extracted data, the same set of training examples was provided to both KDD systems. The only difference between them is the training data for the rule-miner of DISCOTEX is automatically extracted by RAPIER after being trained on a disjoint set of 60 user-labeled examples. Both systems are tested on user-labeled data to identify the quality of the rules produced. Figure 3.6 shows the learning curves for precision and recall.

Even with an extractor trained on a small amount of user-labeled data, the results indicate that DISCOTEX achieves a performance fairly comparable to the rule-miner trained on a manually constructed database, while random-guessing does quite poorly. Figure 3.6 indicates that DISCOTEX

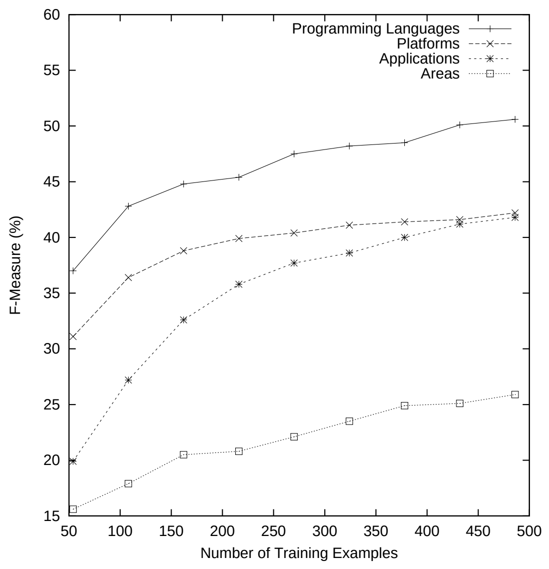


Figure 3.7: F-measure for DiscoTEX by slots

does relatively worse with the first 60 training examples with respect to recall, but quickly improves with 60 additional examples. The results also show that the precision of DiscoTEX seems to start leveling off a bit sooner, this is presumably due to the fact that extraction errors put a somewhat lower ceiling on the performance it can eventually achieve.

Figure 3.7 presents F-measures for DiscoTEX's performance on individual slots. Not surprisingly, the Programming Languages slot with the least number of possible values shows the best performance, and the Area slot with as many as 210 values does poorly. More interesting is the fact that different slots show quite different learning rates.

Figure 3.8 shows the learning curves for precision and recall under the "more natural" scenario in which the training set provided to Rapier, consisting of 60 user-labeled examples, is also provided to the rule-miner as a part of its training set. However, as DiscoTEX proceeds to discover knowledge from data it automatically extracts from raw text, it fairly closely tracks the performance of a system trained on additional data laboriously extracted by a human expert. Since in this case DiscoTEX has the advantage of a small set of relatively noise-free data to start with, its performance is even somewhat closer to that achieved by mining a hand-built database.

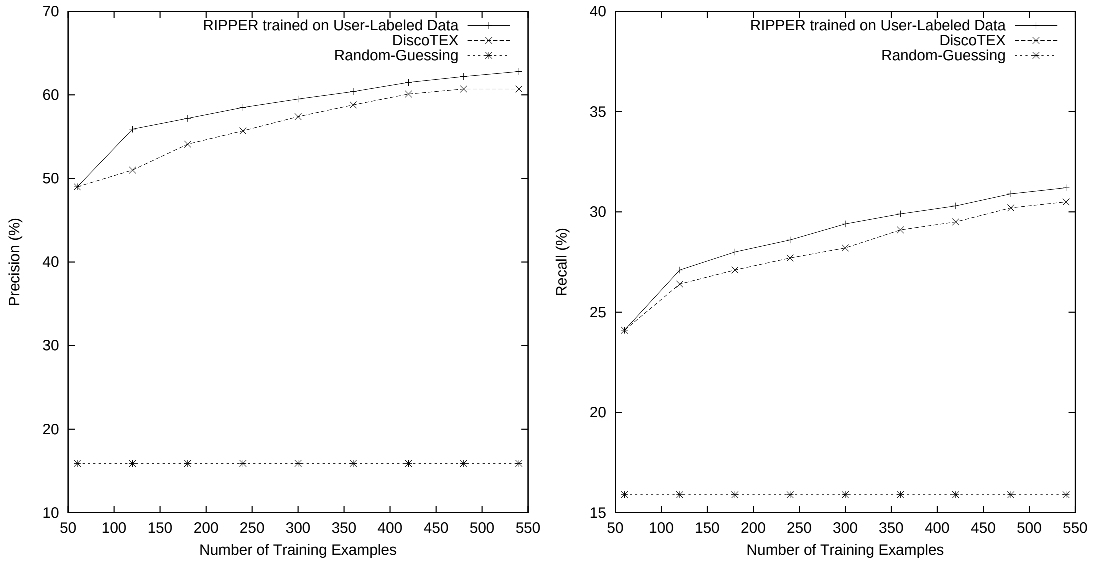


Figure 3.8: Precision and recall with reused IE training set

3.6 Summary

In this chapter we demonstrated that combining IE and KDD is a viable approach to text mining by showing that mined rules from an automatically extracted database are fairly accurate in comparison with those discovered from a manually constructed database. We first presented a framework called DiscoTEX employing an IE module for transforming natural-language documents into structured forms and a KDD module for mining prediction rules. While information retrieval approaches view texts as sets of terms, each of which behaves based on some form of frequency distribution, traditional machine learning approaches view texts as sets of features whose combinations are usually learned by inductive methods. In order to exploit richer information provided by an underyling IE system about the structure of individual documents, we combined traditional ways of representing documents with the feature vector model. Finally, experimental results obtained on a corpus of USENET job postings with an initial implementation of the DiscoTEX framework are presented and discussed.

Chapter 4

TextRISE: Learning Soft-Matching Rules From Text

As discussed in Section 3.5, one step that is performed manually in the initial experiments is collapsing similar slot-fillers in the extracted data into a canonical form. For example, "NT," "Windows NT," and "Microsoft Windows NT" are typically extracted fillers for the platform slot in the USENET job announcement domain. All of those are mapped to a unique term by a synonym-checking dictionary before the rule mining step and treated as the same attribute afterwards. Such collapsing could be automated by clustering slot fillers using a textual similarity metric (Bilenko & Mooney, 2003).

An alternative approach we adopted in this thesis is to allow partial matching of slot-fillers during the discovery process, instead of requiring or creating canonical slot-fillers that must match exactly. In this chapter, we will present an implementation of such a soft-matching rule mining algorithm, called TextRISE (Nahm & Mooney, 2001). In TextRISE, a flexible metric is used to find examples that are close but not exact matches to the conditions of a rule. We consider a problem of predicting a textual slot value such as BOWs, tokens, strings, or numbers for each slot, instead of predicting the presence or absence of a specific slot value like a standard rule learner.

4.1 RISE

The need for soft matching of text strings in discovered rules is an aspect of text mining that requires changes to existing rule induction methods. In this section, we explore the discovery of rules that allow soft matching of slot-fillers by adapting the RISE algorithm of unifying rule-based and instance-based learning methods (Domingos, 1996).

Instance-based learning, or memory-based learning techniques work essentially by keeping typical examples for each class (Aha, Kibler, & Albert, 1991). In general, three characteristics are defined for instance-based learning algorithms: 1) a similarity function telling the algorithm how close two instances are, 2) a typical instance selection function indicating which of the instances are typical or atypical, and 3) a classification function deciding how a new case is related to the learned cases. Instance-based learning algorithms are conceptually simple and easy to test although they suffer from the "incomprehensibility" problem due to not producing concepts in a human readable format. They also sometimes require moderately large amount of storage.

The RISE (Rule Induction from a Set of Exemplars) algorithm was proposed to overcome the well-known *small disjuncts problem* and *splintering problem* of rule induction, and mitigating instance-based learning's vulnerability to noise and irrelevant features at the same time. Unlike other combined model approaches, RISE is a unified single algorithm which is able to behave both as an instance-based classifier and a rule induction system. In extensive experiments, RISE was fairly consistently more accurate than alternative methods, including standard rule-based and instance-based algorithms (Domingos, 1996).

Instead of requiring rules to match exactly in order to make a prediction, RISE makes predictions by selecting the closest matching rule according to a standard distance metric typically used in nearest-neighbor methods (a modified Euclidian distance). By generating generalized rules instead of remembering specific instances and by using a similarity metric rather than exact matching to make predictions, it elegantly combines the properties of rule induction and instance-based learning.

Flexible-matching rules are acquired using a specific-to-general (bottom-up) induction algorithm that starts with maximally specific rules for every example and then repeatedly minimally generalizes each rule to cover the nearest example it does not already cover, unless this results in a decrease in the performance of the overall rule base. Performance of a rule base is measured by conducting leave-one-out testing on the training data using the

```
Input: $ES$ is the training set.
Output: $RS$ is the rule set.

Function RISE ($ES$)

$RS := ES$.
Compute $Accuracy(RS, ES)$.
Repeat
    For each rule $R \in RS$ do
        Find the nearest example $E$ to $R$ (not covered).
        $R' :=$ MostSpecificGeneralization($R$, $E$).
        $RS' := RS$ with $R$ replaced by $R'$.
        If $Accuracy(RS', ES) \geq Accuracy(RS, ES)$
            $RS := RS'$.
        Delete $R'$ from $RS$ if it is a duplicate.
Until no increase in $Accuracy(RS, ES)$ is obtained.
Return $RS$.
```

Figure 4.1: The RISE rule-learning algorithm

closest-matching rule for making predictions. This process repeats until any additional generalization does not increase performance. When classifying test examples, the nearest rule to each example is found, and the rule's class is assigned to the example. The RISE algorithm for learning rules is summarized in Figure 4.1.

Figure 4.2 and Figure 4.3 presents the algorithm for generalizing rules and computing the accuracy of a rule set. In RISE, an example or an instance is simply a rule in which the consequent is the example's class. In the remainder of this chapter, the word "rule" is used to refer both to rules of the general type and the stored examples.

For a walk-through example, consider the following training set for voting records.

```
1) y  y  n  n  y  ->   republican
2) n  y  y  n  n  ->   republican
3) n  n  n  y  n  ->   republican
4) y  n  n  n  n  ->   democrat
5) n  n  y  y  n  ->   democrat
```

These input/output pairs associate patterns of voting decision with party

```
Input: $R = (A_1, A_2, ..., A_n)$ is a rule.
       $E = (E_1, E_2, ..., E_n)$ is an example.
       $A_i$ and $E_i$ are either ? or symbolic values ($r_i$).
Output: $R'$ is the generalized rule.

Function MostSpecificGeneralization ($R$, $E$)

For each attribute $i$ do
    If $A_i$ = ? Then ${R_i}'$ := ?.
    Else if $E_i = r_i$ Then ${R_i}' := R_i$.
    Else if $E_i \neq r_i$ Then ${R_i}'$ := ?.
$R' := ({R_1}', {R_2}', ..., {R_n}')$.
Return $R'$.
```

Figure 4.2: Generalization of a rule to cover an example

```
Input: $ES$ is the example set.
       $RS$ is the rule set.
Output: $Acc$ is the accuracy.

Function Accuracy ($ES$, $RS$)

$Sum := 0$
For each example $E \in ES$ do
    Find the nearest rule $R$ to $E$ (except $R$ s.t. $R = E$).
    $C(R)$ := class label of $R$.
    If $C(R) = C(E)$
        $Sum := Sum + 1$.
$Acc := \frac{Sum}{size(ES)}$.
Return $Acc$.
```

Figure 4.3: The accuracy-computing algorithm

allegiance. The output variable shows whether the politician that cast the votes was Republican or Democrat. Within the input, a 'y' indicates a vote in favor while a 'n' indicates a vote against.

We assume that the Manhattan distance is used to find the closest rule. The initial rule set for the RISE algorithm is the same as the training set since RISE starts with maximally specific rules. The initial accuracy of the rule set is 0 because no example is correctly classified by this rule set with the leave-one out method.

After calculating the initial accuracy, we locate the most similar example for the first rule in the current rule base (among those that have the same class). Example 1) is not considered as a candidate because it is already covered by this rule. The closest example in the training examples is 2, that is generalized with rule 1 in the current rule set. The generalized rule between these two is:

```
1') ?  y  ?  n  ?   ->  republican
```

By replacing rule 1) with $1)'$, the rule set to be compared with the original one is updated as follows.

```
1') ?  y  ?  n  ?   ->  republican
2)  n  y  y  n  n   ->  republican
3)  n  n  n  y  n   ->  republican
4)  y  n  n  n  n   ->  democrat
5)  n  n  y  y  n   ->  democrat
```

We calculate the global accuracy of the updated rule set in order to determine if the old rule should be replaced with the generalized one. Every example in the training set is matched successively to rules in those rule sets. Since the generalized rule increases the accuracy by classifying the second example (which used to be classified incorrectly as `democrat` by the old rule set) correctly, we accept this updated rule base and repeat this process with all the remaining rules. As a result, we get the second-stage rule set like the following (after eliminating duplicated rules). Rule 2 is the generalization of rule 2 (n y y n n $\rightarrow$ republican) and rule 3 (n n n y n $\rightarrow$ repubilcan) in the above rule set while rule 3 is the generalization of rule 4 and rule 5.

```
 1) ?  y  ?  n  ?  ->   republican
 2) n  ?  ?  ?  n  ->   republican
 3) ?  n  ?  ?  n  ->   democrat
```

The algorithm is terminated at this point because no rules can be generalized with any examples in the training set to increase the overall accuracy. Generalizations of rule 1 or rule 2 with the closest training example,

```
    ?  ?  ?  ?  ?  ->   republican
```

cause false predictions (classifying `democract` as `republican`), resulting in a decrease in the global accuracy.

If we are now presented with the following test case,

```
    y  y  y  y  y  ->
```

the output value is `republican` because the closest rule to this test example is rule 1 in the final rule base and its class is `republican`.

Domingos (1996) empirically shows that the RISE algorithm can create synergistic effects between rule induction and instance-based learning. RISE has two major advantages over rule induction. First, RISE is better at dealing with exceptions while rule induction suffers from the small disjuncts problem. Second, it mitigates the splintering problem of having a dwindling number of available examples during the induction process, by evaluating each rule set with respect to the accuracy on the entire training set. Training is reasonably computationally efficient, requiring time $O(e^2, a^2)$ where e is the number of examples, and a the number of attributes.

4.2 The TextRISE Algorithm

RISE is not directly applicable to mining rules from extracted text because: 1) its similarity metric is not text-based and 2) it learns rules for classification rather than text prediction. TextRISE presented in this section addresses both of these issues.

4.2.1 Rule Generalization

First of all, a text-based similarity metric is required to apply RISE to textual data. RISE assumes a Euclidean similarity metric to measure the similarity between two examples. As shown in Section 3.2, a standard vector-space metric from information retrieval (IR) (Baeza-Yates & Ribeiro-Neto, 1999) is used to provide an appropriate similarity metric for TextRISE. For shorter strings, a string edit-distance is employed. Classification accuracy as a measure of performance is replaced with the average similarity of the text predicted to fill a slot and the actual filler.

Computing Generalization

In the BOW model (Section 3.2.2), extracted text is represented as a bag-of-word (BOW), assuming a single slot filler for each slot. To compute the minimal generalization of two BOWs, bag intersection is used. For instance, the generalization of the `title` slot in Figure 3.2 ({“harry”, “potter”, “order”, “phoenix”, “book”}) and that of “Harry Potter and the Goblet of Fire (Book 4)” (`Title` = {“harry”, “potter”, “goblet”, “fire”, “book”}) is a simple intersection of those two, which is `Title` = {“harry”, “potter”, “book”}. The minimal generalization of two examples or rules is the minimal generalization of the BOWs in each of their corresponding slots.

In the string model (Section 3.2.2), strings are represented as sequences of characters. The minimal generalization of two strings is a string that has the same distance to both strings. Figure 4.4 shows the pseudocode for computing the generalization based on the affine gap cost function (See Figure 2.2 for the pseudocode of computing affine gap cost function). The generalization algorithm first computes the distance matrix and then traces back to the point at which the two given strings have the same distances. This algorithm can be applied to the token model (Section 3.2.2) as well.

For example, “Windows” and “Windows 98/2000” are generalized by finding an intermediate string such as “Windows 98/2”. In this case, the distance between “Windows” and “Windows 98/2” must be the same with the distance between “Windows 98/2” and “Windows 98/2000” ($AffineGapCost$ (“Windows”, “Windows 98/2”) = $AffineGapCost$ (“Windows 98/2”, “Windows 98/2000”)). Given “Windows” and “Windows 98/2000”, the algorithm first computes the distance between two strings ($AffineGapCost$ (“Windows”, “Windows 98/2000”)) by repeatedly inserting extra characters such as “ ”, “9”, “8”, “/”, “2”, “0”, “0”, and “0” to the original string, “Windows”. Applied operations are recorded in order. Next, an intermediate string is found by attempting to reconstruct the original string (“Windows”) from the target string (“Windows 98/2000”), applying the recorded operations in a reverse manner. As soon as the distances of the intermediate string to the original and the target strings are equal, the algorithm stops. The same generalization algorithm can be applied to the token model, which represents an extracted slot as a sequence of tokens.

Computing Similarity

RISE finds the nearest example to generalize a given rule, satisfying two requirements: that the example should not be already covered by that rule, and the class assigned to the example is the one predicted by that rule. Since TextRISE does not learn simple categorization rules, the second requirement must be changed: the similarities between the slot-filler of an example and the consequent of the rule should be maximized. To combine

Input: s_1 and s_2 are the given strings.
Output: s_{new} is the generalized string.
Parameters: *match_cost*, *mistmatch_cost*, *gap_start_cost*, *gap_extend_cost*.

```
Function GeneralizationWithAffineGapCost ($s_1$, $s_2$)

$T[m][n]$ := AffineGapCost($s_1$, $s_2$).
$half\_distance := \frac{T[m][n]}{2}$.
$stop$ := no.
For ($i := m$, $j := n$; $i > 0$ and $j > 0$ and $stop = no$; ) do
    If ($s_1[i-1] == s_2[j-1]$) Then $sub\_cost := match\_cost$.
    Else                           $sub\_cost := mismatch\_cost$.
    If (($T[i-1][j-1] + sub\_cost < D[i][j]$) and ($T[i-1][j-1] + sub\_cost < I[i][j]$))
        If ($s_1[i-1] \neq s_2[j-1]$)
          $s_{new}[j-1] := s_1[i-1]$.; $i := i$ - 1.; $j := j$ - 1.
    Else
          If ($D[i][j] < I[i][j]$)
            $T[i][j] := D[i][j]$.
            $len := length(s_{new})$.
            For $k := len$ to $j+1$ do
                $s_{new}[k] := s_{new}[k-1]$.; $s_{new}[j] := s_1[i-1]$.; $i := i$ - 1.
          Else
              $T[i][j] := I[i][j]$
              $len := length(s_{new})$
              For $k := j-1$ to $len - 1$ do
                  $s_{new}[k] := s_{new}[k+1]$.; $j := j$ - 1.
    If ($T[i][j] \leq half\_distance$) Then $stop := yes$
    $len := length(s_{new})$.
    If ($i == 0$)
       For($k := j$;$k > 0$ and $stop = no$;$k--$) do
           $len := len$ - 1.
       If ($T[0][k] \leq half\_distance$) Then $stop := yes$.
    Else if $j = 0$
       For ($k := i$;$k > 0$ and $stop = no$;$k--$) do
           $s_{new}[len] := s_1[i-1]$.; $len := len + 1$.
       If ($T[k][0] \leq half\_distance$ Then $stop := yes$.
Return $s_{new}$.
```

Figure 4.4: The algorithm for computing generalizations of two strings with affine gap cost

```
Inputs: $R = (A_1, A_2, ..., A_n, C_R)$ is a rule
        $E = (E_1, E_2, ..., E_n, C_E)$ is an example.
        $A_i$, $E_i$, $C_R$, and $C_E$ are fillers, possibly empty.
Output: $R'$ is the generalized rule.
Function MostSpecificGeneralization ($R$, $E$)

For $i := 1$ to $n$ do
    If $i$ is using the BOW model
        $A_i' := A_i \cap E_i$.
    Else if $i$ is using the token model or the string model
            $A_i' :=$ GeneralizationWithAffineGapCost$(A_i, E_i)$.
    Else if $i$ is using the numeric (number model)
            $A_i' :=$ NumericalAverage$(A_i, E_i)$.
$R' := (A_1', A_2', ..., A_n'$, Generalization($C_R$, $C_E$)).
Return $R'$.
```

Figure 4.5: Generalization of a rule to cover an example

this requirement with the goal of the original task which is to find the nearest example of a given rule, we calculate the similarity between each example and the given rule to find an example with minimal distance to the rule.

The distance between a rule and an example is formally defined as follows. Let $E = (E_1, E_2, ..., E_n, E_C)$ be an example with E_i for the ith attribute. Let $R = (A_1, A_2, ..., A_n, R_C)$ be a rule. E_C and R_C the consequents of E and R respectively. A_i as well as E_i, E_C, and R_C is either a BOW (BOW Model), a sequence of tokens (Token Model), a string (String Model), or a numeric value (Number Model). The distance $\Delta(R, E)$ between R and E is then defined as:

$$\Delta(R, E) = \sum_{i=1}^{n} \delta(i) + \delta(R_C, E_C) \tag{4.1}$$

where the component distance $\delta(i)$ for the ith attribute is:

$$\delta(i) = \begin{cases} \mathit{cosine\_distance}(R_i, E_i) & \text{if } i \text{ is using the BOW Model} \\ \mathit{affine\_gap\_cost}(R_i, E_i) & \text{if } i \text{ is using the Token Model or the String Model} \\ \delta_{num}(i) & \text{if } i \text{ is using the Number Model} \end{cases} \tag{4.2}$$

where $\mathit{cosine\_distance}(R_i, E_i)$ and $\mathit{affine\_gap\_cost}(R_i, E_i)$ are computed by Equation 2.5.2 and the affine gap cost function described in Figure 2.2

```
Input: $ES$ is the training set.
Output: $RS$ is the rule set.

Function TextRISE ($ES$)

$RS := ES$.
Compute $TextAccuracy(RS, ES)$.
Repeat
        For each rule $R \in RS$ do
            Find the nearest example $E$ to $R$ (not covered).
            $R' :=$ MostSpecificGeneralization($R$, $E$).
            $RS' := RS$ with $R$ replaced by $R'$.
            If $TextAccuracy(RS', ES) \geq TextAccuracy(RS, ES)$
               $RS := RS'$.
            If $R'$ is identical to another rule in $RS$
               delete $R'$ from $RS$.
Until no increase in $TextAccuracy(RS, ES)$ is obtained.
Return $RS$.
```

Figure 4.6: The TextRISE rule-learning algorithm

respectively, and $\delta_{num}(i)$ is the difference in the ith value normalized by its largest observed value:

$$\delta_{num}(i) = \frac{\mid E_i - R_i \mid}{Max_i - Min_i} \tag{4.3}$$

where Max_i and Min_i being respectively the maximum and minium values for the attribute found in the training set. We also define the distance from a missing value to any other as 0. $\delta(R_C, E_C)$ is the distance between R_C and E_C.

4.2.2 The Algorithm

A rule is said to *cover* an example if all of its antecedents are satisfied by the example's corresponding fillers. To extend the algorithm from classification to text prediction, we define a new measure for the accuracy of a rule set on an example set: $TextAccuracy(RS, ES)$ is the average similarity of the predicted fillers for the examples in ES to the corresponding fillers predicted by a rule set RS. The algorithms for generalizing a rule to cover an example

```
Input: $ES$ is the example set.
       $RS$ is the rule set.
Output: $Acc$ is the accuracy.

Function TextAccuracy ($ES$, $RS$)

$Sum := 0$.
For each example $E \in ES$ do
    Find the nearest rule $R$ to $E$ (except $R$ s.t. $R = E$).
    $R_{cons}$ := consequent of $R$.
$Sum := Sum + Similarity(R_{cons}, E_{cons})$.
$Acc := \frac{Sum}{size(ES)}$.
Return $Acc$.
```

Figure 4.7: The accuracy-computing algorithm

and for learning rules are described in Figure 4.5 and Figure 4.6 respectively.

The algorithm is a straightforward modification of RISE using the new similarity and predictive-accuracy metrics and is used to induce soft-matching rules for predicting the filler of each slot given the values of all other slots. The algorithm for computing the accuracy of a rule set is given in Figure 4.7. Unlike RISE, TEXTRISE computes the accuracy of a rule set by accumulating the similarity between the consequent of an example and the consequent of the corresponding rule.

For an example, consider the set for book descriptions in Figure 4.8. Let us assume that all slots are represented as strings except for the "synopsis" slot which is represented as BOWs. We also assume that the "synopsis" slot is to be predicted from the "author" and the "title" slot. As in RISE, the initial rule set for the TEXTRISE algorithm is the same as the training set since TEXTRISE starts with maximally specific rules. When we locate the most similar example with the first rule in the current rule base, we find example 3) is the closest one with example 1). The generalized rule between these two is

```
1')
Author = Isa Asimov
Title  = Norby and the Lost Princess (19
```

$\rightarrow$

```
Synopsis =  {"book", "jeff", "robot", "norby", "rescue", "princess",
```

	Author	Title	Synopsis
1	Isaac Asimov	Norby and the Lost Princess	"In this third book of the Norby series, Jeff and Norby rescue a young princess trapped on a planet."
2	I Asmiov	Lucky Starr	"In this book, Jeff Starr and his partner travel to a planet to investigate accidents and setbacks of a research project."
3	Janet Asimov	Norby and the Lost Princess (1985)	"In this book, Space Cadet Jeff and his robot Norby rescue a princess trapped on another planet."
4	Ursula Le Guin	The Dispossessed	"Shevek, a brilliant physicist, risks his life by traveling to the utopian planet of Urras."

Figure 4.8: A set of book descriptions

```
            "trapped", "planet"}
```

By replacing rule 1) with 1)$'$, the rule set to be compared with the original one is updated as follows.

```
1')
Author = Isa Asimov
Title  = Norby and the Lost Princess (19

→

Synopsis = {"book", "jeff", "robot", "norby", "rescue", "princess",
            "trapped", "planet"}

2)
Author = I. Asimov
Title  = Lucky Starr

→

Synopsis = {"book", "david", "starr", "parner", "travel", "planet",
            "investigate", "accidents", "setbacks", "research", "project"}
```

```
3)
Author = Janet Asimov
Title  = Norby and the Lost Princess (1985)

→

Synopsis = {"book", "space", "cadet", "jeff", "robot", "norby",
            "rescue", "princess", "trapped", "planet"}

4)
Author = Ursula Le Guin
Title  = The Dispossessed

→

Synopsis = {"shevek", "brilliant", "physicist", "risks", "life",
            "traveling", "utopian", "planet", "urras"}
```

We calculate the global accuracy of the updated rule set to see if the old rule should be replaced with the generalized one. Every example in the training set is matched successively to rules in those rule sets. Since the generalized rule increases the accuracy by predicting the consequent slot of the second example more closely, we accept this updated rule base and repeat this process to all the remaining rules. After eliminating duplicates, we obtain the following rule set as a result.

```
1')
Author = Isa Asimov
Title  = Norby and the Lost Princess (19

→

Synopsis = {"book", "jeff", "robot", "norby", "rescue", "princess",
            "trapped", "planet"}

2)
Author = I. Asimov
Title  = Lucky Starr

→

Synopsis = {"book", "david", "starr", "parner", "travel", "planet",
            "investigate", "accidents", "setbacks", "research", "project"}

4)
Author = Ursula Le Guin
Title  = The Dispossessed
```

$\rightarrow$

```
Synopsis = {"shevek", "brilliant", "physicist", "risks", "life",
            "traveling", "utopian", "planet", "urras"}
```

The algorithm terminates at this point because no rules can be generalized with any examples in the training set to increase the overall accuracy. If we are now presented with the following test case:

```
Author =  Janet Asimov
Title  =  Norby and the Court Jester
```

the output value for the subject slot is a BOW, {"book", "jeff", "robot", "norby", "rescue", "princess", "trapped", "planet"} because the closest rule to this test example is rule 1) in the final rule base and it predicts that BOW in the synopsis slot. Sample rules induced from the book descriptions data set (Section 3.3.3) are shown in Figure 4.9.

4.2.3 Interestingness Measures

The output of TEXTRISE is an unordered set of soft-matching rules. Ranking rules based on an interestingness metric can help a human user focus attention on the most promising relationships. Several metrics for evaluating the "interestingness" or "goodness" of mined rules, such as *confidence* and *support*, have been proposed (Bayardo Jr. & Agrawal, 1999).

However, the traditional definitions for confidence and support assume exact matches for conditions. For instance, the support of a rule "C $\in$ languages $\rightarrow$ WindowsNT $\in$ platform" is defined as the number of examples in the database in which `C` $\in$ `languages` and `WindowsNT` $\in$ `platform` occur together. By this definition, an example with `C` $\in$ `languages` and `WinNT` $\in$ `platform` is not counted when support and confidence are computed even though `WindowsNT` and `WinNT` could be treated as a unique item based on the high similarity of those terms. Consequently, we modify the two common metrics, confidence and support, for judging the goodness of the soft-matching rules.

A *rule* consists of two conditions called antecedent and consequent, and is denoted as $A \rightarrow C$ where A is equal to $A_1 \wedge A_2 \wedge ... \wedge A_i$. The *similarity-support* of an antecedent A, denoted as *simsup*(A) is the number of examples in the data set, that are soft-matched by A. In other words, *simsup*(A) is the number of examples to which A is the closest rule in the rule base. The similarity-support of rule $A \rightarrow C$, denoted as *simsup*$(A \rightarrow C)$, is defined as the sum of similarities between C and the consequents of the examples soft-matched by A in the data set. In these definitions, we replace the traditional

title nancy_drew
synopses { nancy(1) }
subject { children(2), fiction(2), mystery(3), detective(3), juvenile(1), espionage(1) }
$\rightarrow$
author keene_carolyn

synopses { role(1), protein(1), absorption(1), metabolism(4), vitamins(1), minerals(1) }
reviews { health(1) }
subject { science(1), human(1), physiology(1) }
$\rightarrow$
title { nutrition(1) }

author beatrice_gormley
synopses { witness(1), ufo(1), landing(1) }
subject { science(1), fiction(2) }
$\rightarrow$
reviews { aliens(1), ufo(1), book(2) }

title charlotte_perk_gilman(1)
synopses { work(1), utopias(1), herland(1), ourland(1) }
reviews { gilman(1), author(1) }
subject { literature(2), criticism(2), classics(1), women(1), literary(1) }
$\rightarrow$
comments { utopia(1), feminist(1) }

Figure 4.9: Sample rules from book data set

hard-matching constraints for a rule with weaker constraints determined relative to all the other rules in the rule base. Similarity-confidence of a rule $A \rightarrow C$, denoted by $simconf(A \rightarrow C)$, is computed as below.

$$simconf(A \rightarrow C) = \frac{simsup(A \rightarrow C)}{simsup(A)}$$

These measures are used to rank the rules generated by TextRISE to show users more interesting rules first. Users can specify the minimum value of similarity support (confidence) for rules to be displayed in order to filter out rules with limited coverages (accuracies) by setting a cutoff level, although a rule pruning mechanism based on minimum confidence or support is not incorporated in the rule learning algorithm of TextRISE as in association rule mining.

4.3 Evaluation

4.3.1 Experimental Methodology

The book data set (Section 3.3.3) is employed in our evaluation of TextRISE. The data set is composed of 6 subsets, science fiction, literary fiction, mystery, romance, science, and children's books. 1,500 titles were randomly selected for each genre to make the total size of the book data set to be 9,000. We used a 6 slots: `titles`, `authors`, `subject` terms, `synopses`, published `reviews`, and customer `comments`. All slots are treated as BOWs.

Unlike a standard rule learner that predicts the presence or absence of a specific slot value, TextRISE predicts a textual value for each slot. Therefore, we evaluate the performance of TextRISE by measuring the average similarity of the predicted slot values to the actual fillers for each slot, e.g. consine similarity for BOW-type slots. We compare the system to a standard nearest-neighbor method to show that TextRISE's compressed rule base is superior at predicting slot-values. In both methods, prediction is made by selecting the closest rule/example using only the text in the antecedent slots. We also tested nearest-neighbor without using information extraction to show the benefit of IE-based text mining. To clearly show IE's role, the only change made to nearest-neighbor was to treat the set of antecedent slots as a single, larger text.

In addition to the textual similarity, we developed analogs for precision and recall. Precision and recall were defined as follows, where C is the correct slot and P is the predicted one.

$$precision = similarity(generalization(C, P), P) \tag{4.4}$$

$$recall = similarity(generalization(C, P), C) \tag{4.5}$$

F-measure is defined as the harmonic mean for precision and recall as previously shown in Equation 3.3. For example, the precision and recall are defined as follows when C and P are BOWs, precision and recall are defined as follows.

$$precision = similarity(C \cap P, P) \tag{4.6}$$

$$recall = similarity(C \cap P, C) \tag{4.7}$$

4.3.2 Results and Discussion

The experiments were performed using ten-fold cross validation. Learning curves for predicting the `title` slot are shown in Figure 4.10. The graph shows 95% confidence intervals for each point. All the results on average similarities, precisions, and F-measures were statistically evaluated by a one-tailed, paired *t*-test. For each training set size, two pairs of systems(TextRISE versus nearest-neighbor and nearest-neighbor versus nearest-neighbor without information extraction) were compared to determine if their differences were statistically significant ($p < 0.05$).

The results indicate that TextRISE does best, while nearest-neighbor without IE does worst. This shows TextRISE successfully summarizes the input data in the form of prediction rules. The rule-compression rate of TextRISE is about 68% on average, which means the number of rules TextRISE produces is 68% of the number of examples originally stored in the initial rule base.

Learning curves for precision and F-measure are presented in Figure 4.11. TextRISE provides higher precision, since the conclusions of many of its rules are generalized slots, and overall F-measure is moderately increased. The average similarity, precision, and F-measures are low (under 25%) because predicting textual slots given the information on other slots is a relative difficult task.

We also conducted the experiments on the movie data set (Section 3.3.4) and obtained similar results as shown in Figure 4.12. The learning curves are for predicting the `title` slot.

4.4 Summary

In this chapter, we showed that instance-based learning and rule-learning algorithms can be integrated to discover soft-matching rules from textual data. Such a hybrid is a good match for text mining since rule-induction provides

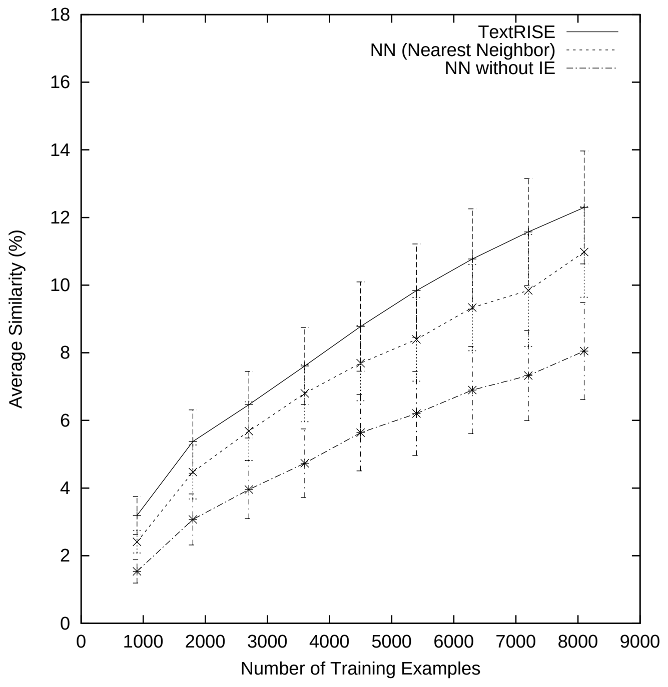
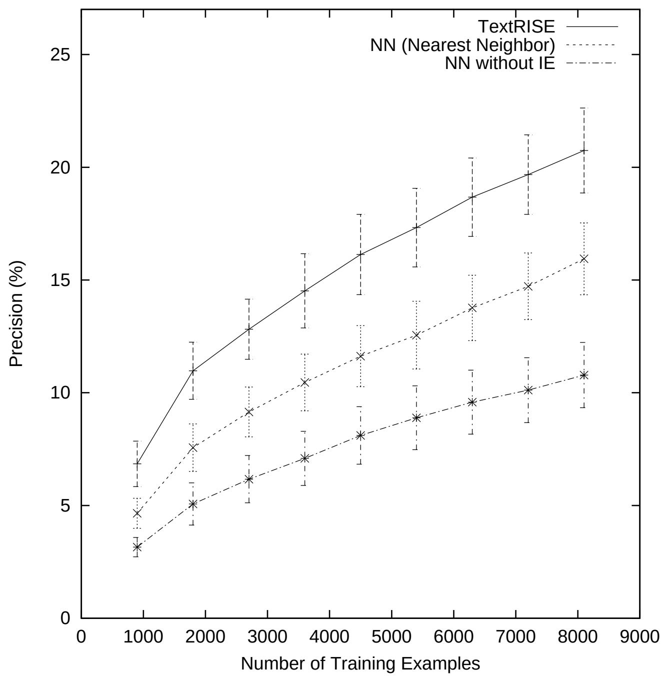


Figure 4.10: Average similarities for book data

simple, interpretable rules, while nearest-neighbor provides soft matching based on a specified similarity metric. In TextRISE, the user gives a similarity metric for each field. Our approach uses a TFIDF text-similarity metric from information retrieval (Baeza-Yates & Ribeiro-Neto, 1999) for long text and a standard edit-distance metric for short strings. Generalization methods for each model, such as bag intersections and intermediate strings, are presented.



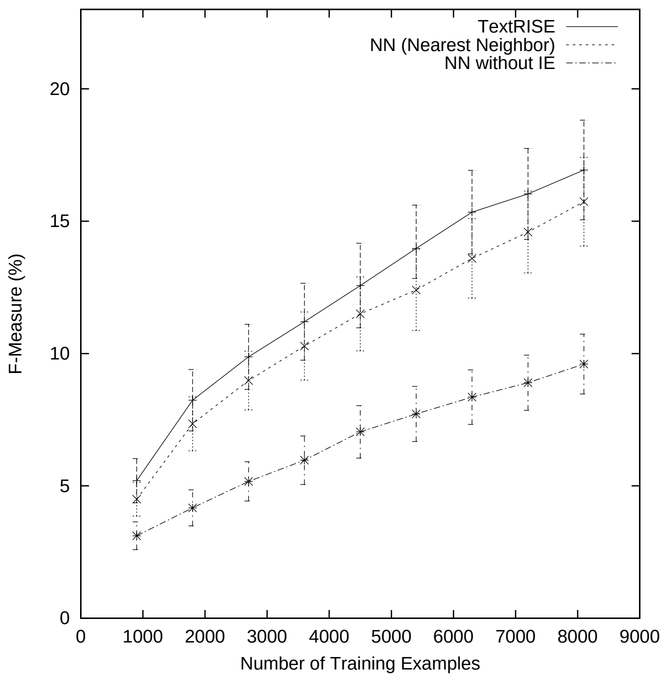


Figure 4.11: Precision and F-measures for book data

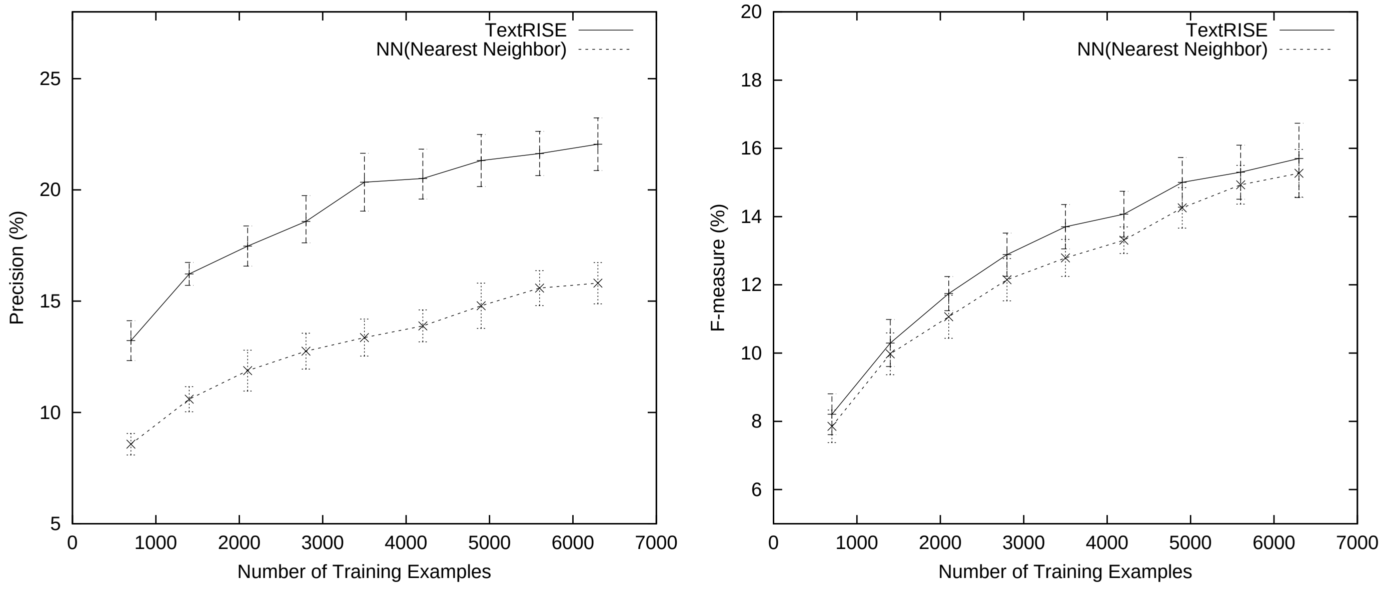


Figure 4.12: Precision and F-measures for movie data

Chapter 5

SoftApriori: Mining Soft-Matching Rules from Text

Association rules mining is one of the most popular methods in data mining (Han & Kamber, 2000; Witten & Frank, 1999) (See Section 2.3.2). By directly applying the standard association rule mining algorithm such as Apriori (Agrawal & Srikant, 1994) to text, associations between the extracted items could be discovered. Documents can be treated as baskets and extracted fillers as items. However, one of the problems of association rule mining techniques is that each item is always considered to be distinct from the others, therefore any two items are either the same or totally different.

As previously stated in Section 1.2, the content of the extracted item may not always be an exact text match with known values. Instead, it may be a close match. The problem of matching these extracted items to the actual known values is called "soft matching". In this chapter, we explore the generalization of the standard algorithm for discovering association rules to allow for soft matching based on a given similarity metric for each field. Soft matching association rules whose antecedents and consequents are evaluated based on sufficient similarity to database entries are able to directly mine "dirty" data by overcoming the barrier of the traditional rule mining that only takes fixed categorical values (or items).

For instance, consider the example in Table 5.1, that lists required skills for a set of computer-science jobs. In this database, the co-occurrence of "DB Management (or Data Management)" in areas, "Windows (or Windows98)"

ID	Areas	Platforms	Applications
1	Data Management	Windows	Microsoft Access
2	DB Management	Windows98	MS Access
3	Web	WindowsNT	VBScript
4	World Wide Web	WinNT	ASP

Table 5.1: Sample noisy textual database

in platforms, and "Microsoft Access (or MS Access)" in applications is a pattern that a human can easily recognize (from jobs 1 and 2). However, traditional rule mining techniques such as the association rule mining algorithm (Section 2.3.2) cannot discover such patterns because they treat "Data Management" / "DB Management", "Windows" / "Windows98", and "Microsoft Access" / "MS Access" as different items. This example motivates a rule-learning algorithm that allows partial matching.

5.1 Soft Association Rules

In this section, we introduce the problem of mining *soft* association rules from databases and investigate how to utilize an existing association rule mining algorithm to incorporate similarity in discovering associations. With a softened definition for associations that does not require exact matches, we present an algorithm called SoftApriori for discovering soft association rules, as well as implementations using a string edit-distance and a cosine similarity as the similarity metrics.

Before presenting our algorithm for discovering soft association rules, we define soft relations as follows. We assume that a function, $similarity(x, y)$, is given for measuring the similarity between two items x and y. The range of the similarity function is the set of real numbers between 0 to 1 inclusive, and $similarity(x, y) = 1$ iff $x = y$.

Definition 1 (is-similar-to) *An item x is similar to an item y ($x \sim y$) iff $similarity(x, y) \geq T$, where T is a predefined threshold between 0 and 1. We also define a binary function $similar(x, y)$ which is 1 if $x \sim y$ and 0 otherwise. This definition is a natural generalization of "x equals to y ($x = y$)" with T set to 1. The similarity relation is reflexive ($x \sim x$) and symmetric ($x \sim y$ implies $y \sim x$), but not transitive($x \sim y$ and $y \sim z$ does not necessarily imply $x \sim z$).*

Record	Items
R_1	a, b, c, d
R_2	a', b', c, d'
R_3	a, c, c', d'
R_4	a'', d, e

Table 5.2: An example of a database with soft-matching items

Definition 2 (is-a-soft-element-of) *An item x is a soft-element of an itemset I ($x \in_{soft} I$) iff there exists an $x' \in I$ such that $x' \sim x$.*

Definition 3 (is-a-soft-subset-of, set-similar) *An itemset I is a soft-subset of an itemset J ($I \subseteq_{soft} J$) iff for every item in I there is a distinct similar item in J, i.e. for every item $x_i \in I$, $I = \{x_1, ..., x_m\}$, there is an item $y_j \in J$ such that $x_i \sim y_i$ and $y_i \neq y_j$ for all $j \neq i$, $1 \leq j \leq m$. Two sets I and J are similar, denoted by $I \sim J$, iff $I \subseteq_{soft} J$ and $J \subseteq_{soft} I$. I is a proper soft-subset of J iff $I \subseteq_{soft} J$ holds but $I \sim J$ is not true.*

Definition 4 (soft-disjoint) *Two itemsets I and J are soft-disjoint when no item in I is a soft-element of J. The "soft-disjoint" relation is symmetric, i.e. if I and J are soft-disjoint then so are J and I.*

For instance, consider the example in Table 5.2. Let us assume that those items with $'$ are similar to items with the same literal without $'$, but not similar to those with other literals. With this assumption, a is similar to a' ($a \sim a'$) and a'' ($a \sim a''$), but not similar to b ($a \not\sim b$). R_1 ($\{a, b, c, d\}$) is a soft-subset of R_2 ($\{a', b', c, d'\}$) since every item in R_1 is similar to some item in R_2. R_2 is also a soft-subset of R_1 and that makes R_1 and R_2 similar to each other. However, R_3 ($\{a, c, c', d'\}$) is not a soft-subset of R_2 since c and c' in R_3 have only one shared similar item c in R_2, but a one-to-one mapping is required for soft-matching items.

The following is a formal statement of the problem of mining soft association rules: Let $I = \{i_1, i_2, ..., i_m\}$ be a set of literals, called items. Let $\mathcal{D}$ be a set of records, where each record R is a set of items such that $R \subseteq I$. A soft association rule is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and X and Y are soft-disjoint. The problem of mining soft association rules is to find all soft association rules, $X \Rightarrow Y$, such that the *soft-support* and the *soft-confidence* of $X \Rightarrow Y$ are greater than the user-defined minimum values (called *minsup* and *minconf* respectively). Formal definitions for soft-support and soft-confidence, which are straightforward generalizations of the traditional ones, are given below.

Definition 5 (soft-support) *The soft-support of an itemset X in a set of records (database) $\mathcal{D}$, denoted as $softsup(X)$, is the number of records, $R \in \mathcal{D}$, such that $X \subseteq_{soft} R$. The soft-support of a rule $X \Rightarrow Y$ in a database $\mathcal{D}$, denoted as $softsup(X \Rightarrow Y)$, is the number of records $R \in \mathcal{D}$ such that $X \cup Y \subseteq_{soft} R$.*

Definition 6 (soft-confidence) *The soft-confidence of a rule $X \Rightarrow Y$, denoted as $softconf(X \Rightarrow Y)$ is given by:*

$$softconf(X \Rightarrow Y) = \frac{softsup(X \Rightarrow Y)}{softsup(X)}$$

For example, the soft-support of the itemset $\{a, c\}$ for the database shown in Table 5.2 is 3 since it is a soft-subset of 3 records, R_1, R_2, and R_3. The soft-confidence of the association rule, $\{a, c\} \Rightarrow \{b'\}$ is computed by dividing the soft-support of $\{a, c, b'\}$ $(= \{a, c\} \cup \{b'\})$ by the soft-support of $\{a, c\}$. Since the soft-support of $\{a, c, b'\}$ is 2 (R_1 and R_2), the soft-confidence of this rule is 2/3, or 66.67%.

5.2 The SoftApriori Algorithm

The problem of discovering soft association rules can be decomposed into three parts as in traditional association rule mining (Agrawal & Srikant, 1994; Srikant & Agrawal, 1995): discovering frequent itemsets, rule generation, and (optional) rule filtering. Here we discuss the first part, finding all frequent itemsets with higher soft-support than the user-specified minimum. Given the frequent itemsets, the Apriori algorithm (Agrawal & Srikant, 1994) can be used to generate rules by simply replacing the confidence measure with soft-confidence.

5.2.1 The Algorithm

In the current algorithm, we add an extra constraint to the definition of similar items that avoids a practical problem. In most applications of mining association rules from textual databases, we do not expect similar items to appear together in the same database record. In other words, even though a single record contains string-valued items that are similar by Definition 1, such items generally refer to different entities. For instance, "ASP" and "JSP" are best considered distinct items despite their similar appearance

Record	Items
R_1	a, b, c, d
R_2	e, f, g
R_3	a, c, h, i
R_4	j, d, f, k

Table 5.3: Sample database

when they both occur in the job skills of a single resume. Based on this intuition, the definition of similar items in the context of a given database is restated as follows.

Definition 7 (is-similar-to (in-the-context-of)) *An item x is similar to an item y in the context of database $\mathcal{D}$ ($x \sim_{\mathcal{D}} y$) iff $similarity(x, y) \geq T$ (where the threshold T is a predefined constant between 0 and 1)* **and** *x and y do not appear together in any record R in $\mathcal{D}$ ($\{x, y\} \not\subseteq R\,(R \in \mathcal{D})$).*

For brevity, in the rest of the paper, we use the shorter notation $\sim$ without specifying the database $\mathcal{D}$ when the relevant database is clear from context.

To discover frequent itemsets for soft association rules, we generalize the existing itemset mining algorithm presented by Agrawal and Srikant (1994) in a straightforward way. Since the notion of equality in the traditional definition of an association rule is replaced by similarity, we need to compute the soft-support of each item and itemset by Definition 5. Similarity between items is computed once and cached for future references. In this approach, frequent itemsets under the definition of soft-support (Definition 5) are treated as normal items and the standard Apriori algorithm can be used with minor modifications. Figure 5.1 gives pseudocode for the SoftApriori algorithm. Notations such as L_k (set of frequent k-itemsets) and C_k (set of candidate k-itemsets) are from Agrawal and Srikant (1994).

The first step of the algorithm determines the frequent 1-itemsets. We assume the minimum soft-support value, *minsup*, is provided by the user. The set of frequent 1-itemsets L_1 in SoftApriori is defined as follows:

$$L_1 = \{\{x\} \mid x \in I \land softsup_I(\{x\}) \geq minsup\}$$

In other words, L_1 is the set of all 1-itemsets whose soft-support is greater than the user-given minimum support. By Definition 5, the soft-support of each item is calculated by summing the number of occurrences of all similar items. Formally, the soft-support of a 1-itemset $\{x\}$ where x is an element of the set of all items I ($x \in I$) is computed as follows:

```
Input: $\mathcal{D}$ is the set of records.
Output: $L_k$ is the frequent $k$-itemsets.

Function SoftApriori ($\mathcal{D}$)

$L_1$ := FindFrequentItemsets($\mathcal{D}$).
$k := 2$.
while ($L_{k-1} \neq \emptyset$) do
begin
        $C_k$ := GenerateCandidates($L_{k-1}$).
        forall records $r \in \mathcal{D}$ do
            forall $c \in C_k$ do
                if $c \subseteq_{soft} r$
                    $c.count := c.count + 1$.
        $L_k$ := All candidates in $C_k$ with minimum softsups.
        $k := k + 1$.
end
Return $\bigcup_k L_k$.
```

Figure 5.1: The SoftApriori algorithm

$$softsup_I(\{x\}) = \sum_{y \in I} similar(x,y) \times support(y)$$

While counting the occurrences of all items, we measure the similarity of every pair of items and construct an $m \times m$ matrix $similar(i,j)$, where m is the total number of items in the database. Usually the similarity matrix is extremely sparse since most items are not similar. A hash table is used to store the sparse similarity matrix. Table 5.4 shows a list of the pairs of similar items given in the database in Table 5.3.

To determine frequent 1-itemsets, the soft-supports of all items are computed. Intuitively, we construct a cluster of items containing the items similar to each given "central" item, and sum the support of all items in the cluster. The similarity hash table is used to efficiently retrieve similar items. Table 5.5 shows the items from the sample database in Table 5.3 sorted by decreasing soft-support. Items whose support is less than the minimum support are discarded since they are not frequent.

After constructing sets of frequent similar items, they are treated the same as items in the original Apriori algorithm. Note that the closure property on which the original Apriori algorithm is based still holds for

Similar Items ($x \sim y$)
a, e
a, j
b, f
c, f
d, i
g, i
h, j
Otherwise, $x \not\sim y$

Table 5.4: Similar items

Item (Similar Items)	Support	Frequent
a (e, j)	4	✓
f (b, c)	4	✓
i (d, g)	4	✓
b (f)	3	✓
c (f)	3	✓
d (i)	3	✓
e (a)	3	✓
j (a, h)	3	✓
g (i)	2	
h (j)	2	
k	1	

Table 5.5: Sample frequent 1-itemset table (minsup = 3)

soft itemsets. In other words, if an itemset has a soft-support higher than minsup, then every subset of that itemset also has soft-support higher than minsup. Given L_{k-1}, the set of all frequent $(k-1)$-itemsets, the candidate itemsets C_k are generated by self-joining L_{k-1} with L_{k-1}.

In a manner similar to the initial construction of frequent items, itemsets are grown by computing the soft-support of candidates and discarding those with low soft-support. The soft-subset function is used to check which itemsets in C_k are *softly* in record r. For each itemset that is a soft-subset of r, or for each set of items that have similar items in r, the soft-support of k-itemset is again computed by the equation in Definition 5, counting the number of soft-matching items, instead of simply counting the number of occurrences of each item.

For example, consider the set for the required skills for job announcements shown in Figure 5.1. The data set can be translated into a database, or baskets of items like following:

```
 1) area_Data_Management, platform_Windows, application_Microsoft_Access
 2) area_DB_Management, platform_Windows98, application_MS_Access
 3) area_Web, platform_WindowsNT, application_VBScript
 4) platform_WinNT, application_ASP
```

Let us assume that the minimum soft-support value is set to 2. In the traditional association rule mining algorithm, no frequent 1-itemset can be found from this data set because there is no pair of identical items. However, by considering *similar* items such as "area_Data_Management" and "area_DB_Management" and "platform_Windows98" and "platform_WindowsNT", SoftApriori is able to capture frequent 1-itemsets as:

```
              Frequent 1-itemset                            Soft-Support
{platform_Windows (platform_Windows98, platform_WindowsNT)}     3
{platform_WindowsNT (platform_Windows, platform_WinNT)}         3
{area_Data_Management (area_DB_Management)}                     2
{application_Microsoft_Access (application_MS_Access)}          2
```

therefore allowing the algorithm to grow itemsets further. In the next step, frequent 2-itemsets are identified:

```
              Frequent 2-itemset                            Soft-Support
{platform_Windows (platform_Windows98, platform_WindowsNT),     2
 application_Microsoft_Access (application_MS_Access)}
{platform_WindowsNT (platform_Windows, platform_WinNT),         2
 area_Data_Management (area_DB_Management)}
{application_Microsoft_Access (application_MS_Access),          2
 area_Data_Management (area_DB_Management)}
```

Frequent 3-itemsets are again found as follows, and the algorithm stops here since no frequent 4-itemsets can be found.

```
              Frequent 3-itemset                            Soft-Support
{platform_Windows (platform_Windows98, platform_WindowsNT),
 application_Microsoft_Access (application_MS_Access),          2
 area_Data_Management (area_DB_Management)}
```

From the above frequent itemsets, rules are generated in a straightforward way: For each frequent itemset L, all nonempty subsets are generated first. For every nonempty subset S, the rule, "$S \Rightarrow (L - S)$" is

Job-postings

1. database (databases, database sys.) ∈ area ⇒ oracle (oracle7) ∈ application [3.2%, 43.2%]
2. mfc ∈ area ⇒ windows (windows nt, windows 95, windows 3.1, windows 3.x, windowsnt, windows95, windows'95) ∈ platform [2.7%, 39.0%]
3. bsee (bs/ee) ∈ required-degree ⇒ bscs (bs/cs) ∈ required-degree [2.5%, 75.0%]

Resumés

1. unix ∈ programming-language ⇒ visual basic (visual basic 5.0, visual basic 6.0, visual basic 4.0, ms visual basic, visual basic 4, visual basic 5/6) ∈ programming-language [13.0%, 31.2%]
2. netscape (netscape 4.7, netscape 4.x, netscape 6, netscape ldap) ∈ application ⇒ tcp/ip (tcp ip, tcpip) ∈ application [3.3%, 34.5%]
3. c++ (vc++) ∈ programming-language **and** asp (asps) ∈ language **and** unix ∈ platform ⇒ java (java 2, java2) ∈ programming-language [2.3%, 63.6%]

Figure 5.2: Sample discovered soft-association rules

returned if $soft - support(L)/soft - support(S)$ is greater than or equal to the mininum confidence. For instance, "platform_Windows ⇒ application_Microsoft_Access" is returned if:

$$\frac{soft\text{–}support(\{platform\_Windows, application\_Microsoft\_Access\})}{soft\text{–}support(\{platform\_Windows\})} = \frac{2}{3} \geq minconf$$

Sample rules discovered from the job-postings data set (Section 3.3.1) and the resumé data set (Section 3.3.2) are shown in Figure 5.2. Items similar to a given item are shown in parentheses and values for soft-support and soft-confidence are shown in brackets.

5.2.2 Time Complexity

In the original Apriori algorithm, the identification of the frequent itemsets is known to be computationally expensive, having exponential worst-case behavior in $|I|$ (the number of literals) (Agrawal & Srikant, 1994). However, the number of itemsets considered is greatly reduced in practice. Once all sets of frequent itemsets are obtained, it is straightforward to find association rules without scanning the data again. SoftApriori also has the same property in generating candidates and finding rules but requires a pre-processing step of computing similarities between items.

The extra complexity of constructing a similarity matrix in the initial stage is $O(m^2)$ where m is the total number of items since we need to compute the similarity of every pair of items. However, this complexity can be reduced in practice because items in different fields do not need to be compared. By treating every pair of items in different fields as non-similar, we are able to lower the number of similarity computations to $\sum_{k=1}^{N} {m_k}^2$ whereas N is the number of fields and m_k is the number of items in field k.

Depending on the particular similarity metric, additional optimizations are possible. For example, items in numeric fields can be sorted and then similar items can be quickly determined by checking neighboring items in order of proximity until the similarity threshold is exceeded. We present additional optimizations for string edit-distance and cosine similarity in Chapter 6 to reduce the $O(m^2)$ time complexity in finding all similar pairs.

5.3 Evaluation

5.3.1 Experimental Methodology

To determine the accuracy of a set of association rules, we measured precision and recall with respect to predicting the presence of items in a record from other items in that record. We measured the ability of both hard and soft association rules mined from the same training data with the same minimum confidence and support parameters to make accurate predictions on the same disjoint set of test data.

Precision is the percentage of predicted items that are actually present and *recall* is the percentage of actual items that are correctly predicted. We also report *F-measure* which is the harmonic mean of recall and precision (Equation 3.3). If both soft-precision and recall are 100%, then the results are completely correct. Lower precision indicates that the system is producing spurious rules. Lower recall indicates that the system is failing to predict correct slots.

A prediction is judged to be correct iff there is an item in the record that is at least similar to the predicted item (i.e. $\mathit{similarity}(x, y) \geq T$). Antecedents of hard rules are matched using the appropriate hard match-

Domain	Rule	Precision	Recall	F-measure
Job	Soft	89.44	8.68	15.82
	Hard	86.92	8.55	15.57
Resume	Soft	89.45	3.13	6.06
	Hard	69.75	1.92	3.73
Books	Soft	88.47	10.55	19.06
	Hard	66.67	0.32	0.63

Table 5.6: Test accuracies of soft vs. hard association rules (%)

ing criteria and soft rules are matched using the appropriate soft-matching criteria; however, predictions are always judged "softly" in order not to give soft rules an unfair advantage. The pseudocode for the evaluation method is presented in Figure 5.3.

5.3.2 Results and Discussion

The experimental results obtained for the four textual databases are summarized in Table 5.6. This table gives average prediction accuracies for hard and soft association rules using a minimum support and confidence of 10% and 70% respectively for USENET postings and 2% and 70% for book descriptions, and using a similarity threshold of 0.7 for every field. Minimum support for book data (Section 3.3.3) is lower since otherwise no rules at all are found from this data. The results show that the accuracy of soft rules is consistently, significantly higher than that of hard rules. Training accuracy, measured by training and testing on the same entire dataset, shows similar patterns.

We also performed the same experiments while varying these parameters on the resumé data set (Section 3.3.2) as shown in Figure 5.7. Differences for hard and soft rules were evaluated by a two-tailed, paired t-test to determine if they were statistically significant ($p < 0.05$). Overall, the results clearly show that soft rules are generally better than hard rules at discovering reliable regularities in "dirty" data.

5.4 Summary

Tradtional association rule mining methods require terms in discovered rules to exactly match database entries. Normal variation in data items can therefore prevent the discovery of important and interesting relationships. In this

```
Input: $\mathcal{D}_{test}$ (test database), $Rules$ (association rule set)
Output: ($precision$, $recall$)

Function ComputeAccuracy ($\mathcal{D}_{test}$, $Rules$)

$fired := 0.$
$matched := 0.$
$item := 0.$
$predicted := 0.$
for each record $R \in \mathcal{D}_{test}$ do
  /* precision */
  for each $r$ $(A \Rightarrow c) \in Rules$ do
    if (($r$ is hard and $A \subseteq R$) or
        ($r$ is soft and $A \subseteq_{soft} R$))
    then if $r$ is hard then $A' := A$.
                      else $A' := X$ s.t. $X \subseteq R$ and $X \sim A$.
         $fired := fired + 1.$
         if $c \in_{soft} R - A'$
         then $matched := matched + 1.$
  /* recall */
  for each $c' \in R$ do
    $item := item + 1.$
    if there exists a $r$ $(A \Rightarrow c) \in Rules$ s.t.
      $c \sim c'$ and (($r$ is hard and $A \subseteq R - \{c'\}$) or
                  ($r$ is soft and $A \subseteq_{soft} R - \{c'\}$))
    then $predicted := predicted + 1.$
return ($matched/fired$, $predicted/item$).
```

Figure 5.3: Evaluation algorithm for soft-matching association rules

Minconf	Minsup (%)			
(%)	Rule	5	10	15
50	Soft	90.86/3.17	86.95/3.14	84.55/3.13
	Hard	62.19/3.01	60.41/2.76	60.32/2.31
60	Soft	90.79/3.18	87.71/3.13	85.64/3.13
	Hard	66.64/2.89	64.47/2.50	62.16/2.09
70	Soft	91.34/3.18	89.45/3.13	85.76/3.08
	Hard	71.51/2.61	69.75/1.92	74.50/1.43
80	Soft	92.14/3.15	88.37/3.11	84.13/2.82
	Hard	78.84/2.25	79.05/1.46	80.60/0.69

Table 5.7: Test accuracies of soft vs. hard rules

chapter, we presented the SoftApriori algorithm to discover "soft matching" rules that are evaluated using a specified similarity metric. SoftApriori introduces soft-matching to capture additional relationships. Allowing the discovery of soft-matching rules can eliminate the need for certain types of tedious data cleaning prior to knowledge discovery. Compared to TextRISE, an inductive method for learning soft-matching prediction rules presented in Chapter 4, SoftApriori finds *all* association rules with a given soft-support and soft-confidence, and therefore typically discovers a larger set of regularities.

Chapter 6

Retrieving Similar Textual Items Efficiently

We introduced a problem of mining similarity-based rules for text mining in the previous chapters. So far we have been focusing on mining *accurate* rules from texts. However, it is also an important problem to discover rules *efficiently* to build a scalable text-mining system. One of the major bottlenecks in our systems as well as many other text-mining systems is the computational complexity for retrieving *similar* textual items. In this chapter, we show that similarity-based rule mining systems that deal with large amounts of text data can be scaled up by showing how to efficiently retrieve similar items in textual databases.

6.1 Introduction

A practical text mining system needs to be scalable and efficient in terms of time. Table 6.1 shows that there are four problems to be solved with regard to the performance issue. For mining soft association rules in SoftApriori, we cluster similar items within some boundaries. In other words, all items that exceed a pre-determined similarity threshold value must be retrieved. Similarities are measured by edit-distance between two items for shorter strings, while the vector space model and cosine similary are adopted for longer documents. On the other hand, for learning soft prediction rules in TextRISE k-nearest neighbors are used in generalizing similar items, where k is usually 1. We also separate this problem into two categories, one with edit-distance and the other with cosine similarity.

Without optimization, the time complexity of the naive algorithms for

	SoftApriori (Chapter 5)	TextRISE (Chapter 4)
Short String	Edit-Distance Threshold	Edit-Distance k-Nearest Neighbor
Long Documents	Cosine Similarity Threshold	Cosine Similarity k-Nearest Neighbor

Table 6.1: Problem definition

solving these problems is quadratic ($O(n^2)$) in the number of textual items n since all the items need to be compared with every other item. In the realistic situation where n is very large, an $O(n^2)$ algorithm does not scale well. In this chapter, we show that one can provide an optimized algorithm to yield near linear time complexity on realistic, large data sets.

6.2 Fast Retrieval of Similar Strings

6.2.1 Retrieving Similar Strings Using a Threshold

In this subsection, we first present the implementation of a string retrieval system using an edit-distance function and then describe two optimization methods for fast retrieval of similar strings with a given similarity threshold T ($0 \leq T \leq 1$). The problem is to fill out the $n \times n$ similarity matrix for any n strings with either the similarity value of each pair of strings (when the value exceeds the given threshold) or 0 (otherwise).

Optimization I: Using String-Length Information

To measure similarities of string-valued items, a form of edit-distance was adopted as discussed in Section 2.5. In our implementation, we used affine gap cost (Needleman & Wunsch, 1970; Monge & Elkan, 1996). The computation of edit-distance is performed by dynamic programming in time $O(nm)$ when n and m are the lengths of the two input strings. However, the edit-distance computation in our implementation does not always require the full $O(nm)$ time because it stops as soon as the intermediate result exceeds the minimum value computed from the given similarity threshold, T.

The normalized edit-distance of affine gap cost for two strings is defined as shown in Equation 2.2. Without loss of generality, we assume that $|x| \leq |y|$ where $|x|$ ($|x| \geq 1$) and $|y|$ ($|y| \geq 1$) are the lengths x and y in the rest of this chapter. The maximum distance between x and y can be calculated based on their lengths as follows:

$$maximum\_distance(x,y) = \begin{cases} |y| + gap\_cost & \text{if } |x| = 1; \\ mismatch\_cost \times |y| & \text{if } 1 < |x| < gap\_cost \text{ and } |x| = |y|; \\ |x| + |y| + gap\_cost & \text{otherwise.} \end{cases} \tag{6.1}$$

where *gap_cost* is $gap\_start\_cost + gap\_extend\_cost$.

Intuitively, the maximum distance is the distance of two strings which do not share any characters. Let us assume that the mismatch cost, gap start cost and gap extend cost, which are parameters of affine gap cost, are set to 3, 3 and 1 respectively. Here are examples for each of the three cases. 1) "a" and "bcde": 3(mismatch) + 3(gap start) + 1(gap continued) + 1(gap continued) = 8, 2) "ab" and "cd": 3(mismatch) + 3(mismatch) = 6 , 3) "abcd" and "efghij": 3(gap start) + 1(gap continued) + 1(gap continued) + 1(gap continued) + 3(gap start) + 1(gap continued) = 14.

Given a particular edit-distance function, we can reduce the time complexity of determining similar items under a given threshold T. Since edit-distance counts the number of operations needed to change one string to another, two strings cannot be similar if their lengths are too different. For example, we do not have to compute the actual affine gap cost for "isaac-asimov" and "clark" when $T = 0.7$ to confirm they are different because the gap between any 12-character string and any 5-character string is too big to result in a similarity greater than 0.7.

To generalize this observation, we present a proposition to show that there exists an upper bound for the maximum similarity of two strings that only depends on the lengths of the two strings. If that upper bound is less than T then two strings *cannot* be similar.

Proposition 1 *There exists an upper bound for the maximum similarity of two strings x and y where $|x| > 0$, $|y| > 0$, and $|x| \leq |y|$.*

Proposition 1 can be proved by showing that we can derive a function of $|x|$ and $|y|$ for a lower bound of $minimum\_distance(x, y)$. Intuitively, two strings x and y are most similar when they share as many characters as possible, i.e. x is a substring of y in this case since x is always shorter than or of equal size to y. Among all the cases of x being a substring of y, x and y are most similar when x is the starting substring of y (or equivalently, the ending substring of y), e.g. "abc" and "abcde". By the definition of the affine gap function, we add the gap start cost for the first gap then the gap extend cost is accumulated as the gap is increased. For example, the distance between "abc" and "abcd" is 3 while *distance*("abc", "abcde") is 4 and *distance*("abc", "abcdef") is 5, and so on when *gap_start_cost* and *gap_extend_cost* are set to 3 and 1 respectively. Based on such observation, we derive a function for computing the minimum distance of x and y as follows when we assume that $|x| \neq |y|$:

$$minimum\_distance(x, y) = |y| - |x| + (gap\_start\_cost - gap\_extend\_cost) \tag{6.2}$$

It is clear that if the upper bound for the maximum similarity of x and y is not greater than or equal to T, then x and y cannot be similar. In that case, it is redundant to explicitly compute the similarity between x an y to decide if they are similar or not.

By combining this equation for the minimum distance between x and y with Proposition 1 and the definition of similarity in Equation 2.1, we can obtain the following formula for determining if two strings cannot be similar under affine gap cost.

$$\begin{cases} 1 - \frac{|y|+1}{|y|+(gap\_start\_cost+gap\_extend\_cost)} < T & \text{if } |x| = 1 \text{ and } |x| \neq |y|; \\ 1 - \frac{|y|-|x|+(gap\_start\_cost-gap\_extend\_cost))}{|x|+|y|+(gap\_start\_cost+gap\_extend\_cost)} < T & \text{otherwise } (|x| \neq |y|). \end{cases} \tag{6.3}$$

Using this test, we are able to eliminate edit-distance computations for very different strings. The pseudocode shown in Figure 6.1 describes the procedure of applying the filter.

Optimization II: Using a Trigram Index

We can reduce the number of total comparisons between items even further by using an n-gram index (Navarro & Baeza-Yates, 1998). An n-gram is a substring of length n of a given string. It is easy to show that a string x cannot be similar to y, for any reasonably high threshold T, if they do not share a common substring. In addition, a string x cannot be similar to y for a given threshold if they do not share at least k ($k \geq 0$) n-grams. For example, two similar strings "<u>sci</u>ence" and "<u>sci</u>-fi" ($affine\_gap\_cost$("science", "sci-fi") $= 11$) share 1 trigram, e.g. "sci". For any given string x, one can retrieve a list of strings worth comparing by determining the minimum number of n-grams of x that must be shared with any similar string y.

In our implementation, we used a trigram index to efficiently retrieve a list of candidate similar strings for each string. Trigram methods have been shown to be useful for identifying phrases that have a high probability of being synonyms in many tasks such as DNA sequence analysis (McCray &

```
Input: $s$ is the given string.
       $S$ is the set of strings, $s_1$, $s_2$, ..., $s_n$.
Output: $S'$ is the set of strings similar to $s$. ($S' \subseteq S$)
Parameter: $T$ is the similarity threshold.

Function RetrieveUsingStringLength ($s$, $S$)

$S' := \emptyset$.
$x := length(s)$.
For each string $s_i \in S'$ do
    $y := length(s_i)$.
    If $LengthFilter(x, y, T) = True$
        $sim :=$ similarity($x$,$y$).
        If $sim \geq T$
            $S' := S' \cup \{s_i\}$.
Return $S'$.

SubFunction LengthFilter ($x$, $y$, $T$)
If $x = y$
    Return $False$.
If $x > y$ swap($x$,$y$).
If $x = 1$.
    $maximum\_similarity$
    $:= 1 - (|y| + 1) / (|y| + (gap\_start\_cost + gap\_extend\_cost))$.
Else
    $maximum\_similarity$
    $:= 1 -$
        $(|y| - |x| + (gap\_start\_cost - gap\_extend\_cost))/(|x| + |y| + (gap\_start\_cost + gap\_extend\_cost))$.
If $maximum\_similarity \geq T$
    Return $True$.
Else
    Return $False$.
```

Figure 6.1: The optimization algorithm using string length

	$\|x\| = 1$	$\|x\| = 2$	$\|x\| = 3$	$\|x\| = 4$	$\|x\| = 5$	$\|x\| = 6$	$\|x\| = 7$	...
$\|y\| = 1$	**0**	-	-	-	-	-	-	...
$\|y\| = 2$	3	**0**	-	-	-	-	-	...
$\|y\| = 3$	4	3	**3**	-	-	-	-	...
$\|y\| = 4$	5	4	3	**3**	-	-	-	...
$\|y\| = 5$	6	5	4	6	**3**	-	-	...
$\|y\| = 6$	7	6	5	7	6	**6**	-	...
$\|y\| = 7$	8	7	6	8	7	7	**6**	...
...	...	...	...	...	...	-	-	...

Table 6.2: Minimum distance between x and y with no shared trigrams

Aronson, 2002) or automatic spelling correction (Angell, Freund, & Willet, 1983). Each string is indexed under every three-character substring that it contains. For example, "science" is indexed by 5 trigrams, "sci", "cie' , "ien", "enc" and "nce".

To compute the minimum number of trigrams to be shared by similar items, we first present a proposition which states that there exists a minimum number of trigrams required to be shared by any pair of similar strings.

Proposition 2 *There exists an integer $k \geq 0$ such that any similar pair of strings x and y share at least k trigrams.*

Proposition 2 can be restated: there exists an integer $k \geq 0$ such that no y which shares only k' ($k' < k$) trigrams with x can be similar to x. Note that our task is to find a minimum number k for retrieving all y's that have a possibility of being simliar to the given x when y has at least k shared trigrams with x.

Before showing how to prove the above proposition, let us compute the minimum distance between x and y when they share no trigrams. In that case, the distance of x and y is minimized when they share as many bigrams as possible. For example, assume that x is "$\underline{\text{ab}}\text{c}\underline{\text{de}}$" and y is "$\underline{\text{ab}}\text{f}\underline{\text{de}}$" so that x and y do not share any trigram. The distance between x and y is minimized since there is only one mismatch ("c" and "f") between the bigrams they share ("ab" and "de"). By repeating this computation, we obtain a table shown in Table 6.2 for each $|x|$ and $|y|$ when the mismatch cost, gap start cost, and gap extend cost, which are parameters of affine gap cost, are set to 3, 3 and 1 respectively. These are commonly used values for the penalties (Nahm et al., 2002).

We can easily see from the Table 6.2 that 1) $|x|$ and $|y|$ should be the same to make x and y most similar and that 2) the minimum distance between x and y (x and y share no trigrams) can be written as $3 \times \lfloor |x|/3 \rfloor$.

To prove the Proposition 2, we need to show that the maximum similarity of x and y is always less than T for some $k \geq 0$ when k is the number of trigrams shared by x and y. In other words, it has to be shown that the the minimum distance between x and y is always greater than $1 - T$ for some k. So our goal is to find the lower bound of the *minimum_distance*(x, y) when it is known that x and y share k trigrams and check whether that lower bound exceeds $1 - T$ or not.

The distance between x and y is minimized when the shared portion of x and y is as great as possible. In other words, x and y should share either $3k$ characters if $3k \geq |x|$ or $|x|$ characters otherwise. The distance is minimized especially when the shared characters in two strings are aligned consecutively, e.g. "$\underline{\text{abcdef}}$g" and "$\underline{\text{abcdef}}$hi" where two trigrams "abc" and "def" are shared. We can assume that they follow our previous observation shown in Table 6.2 since the rest of two strings do not share any trigrams. One thing different from the normal cases is that the first characters of the non-trigram-shared parts of two strings should be different as "g" and "h" in the previous example since otherwise there will be the $(k + 1)$-th shared trigram. So the minimum distance between x and y are written as follows when x and y are greater than 3 for $k > 0$:

$$minimum\_distance(x, y) = \begin{cases} 0 & \text{if } 3k \geq |x|; \\ 3 + 3 \times \lfloor \frac{|x| - 3 \times k - 1}{3} \rfloor & \text{otherwise.} \end{cases} \tag{6.4}$$

It is easy to see that if $|x|$ is 1 or 2 then there is no way for x to share any trigrams with other strings. For $|x| = 3$ and $|y| = 3$, the minimum distance between x and y is 3 when they share no trigrams. By computing the maximum similarity of x and y in that case, we obtain a formula such as $1 - \frac{1}{3} = \frac{2}{3} > T$. Only when the threshold T passses this test, we retrieve all strings y which shares no trigrams with x ($|x| = 3$) to compare with x. Otherwise, we only consider a string x when $|x|$ is greater than 3.

However, the Equation 6.4 does not cover every possible case because of the constraint on k that k should not be 0. We first need to find the range of $|x|$ to which the Equation 6.4 can be applied. In other words, for some sufficiently small $|x|$ and some low T, it is not necessarily true that x cannot be similar to y even though they share no trigrams($k = 0$), e.g. "$\underline{\text{ab}}$c$\underline{\text{de}}$" and "$\underline{\text{ab}}$f$\underline{\text{de}}$". Let's assume that $|x|$ is at least 3 since we can simply exclude the cases for $|x| = 1$ or $|x| = 2$ as mentioned previously. The minimum distance between x and y when they have no common trigrams is $3 \times \lfloor |x|/3 \rfloor$ as described above. If the similarity of x and y in such case, which can be computed by the Equation 2.1 is always less than or equal to T, then the Equation 6.4 cannot be applied to those x's. For example, assume that T is 0.7. The filtering process defined by the computation that follows should be applied to only to x which has $|x| > 5$. For instance, the similarity between "abcdef" and "abgdeh" which share two bigrams is 0.625 which is less than $T = 0.7$.

Back to the equation 6.4, the equation holds when two strings x and y share $3k$ characters aligned consecutively at either the starting or the ending position of the longer string(y). If $|x|$ is sufficiently small, then the minimum distance between x and y is 0 because x might be a substring of y. Otherwise, we assume that x is a starting (or ending) substring of y and the rest of x and y share as many as bigrams. We can define an equation for checking if the maximum similarity of x and y are less than T by Equation 2.1 and Equation 6.1.

$$\begin{cases} 1 < T & \text{if } 3k \geq |x|; \\ \frac{3+3\times\lfloor\frac{|x|-3\times k-1}{3}\rfloor}{2\times|x|+4} > 1-T & \text{otherwise.} \end{cases} \tag{6.5}$$

From the first condition which is always false since T is between 0 and 1, we obtain a minimum value for k which is $\lfloor\frac{|x|}{3} - 1\rfloor$. The second part of the above equation is rewritten as follows by separating three cases by computing $|x|\ mod\ 3$.

$$\begin{cases} k < (2\times T-1)\times|x| + (4\times T+2) & \text{if } |x|\ mod\ 3 = 0; \\ k < (2\times T-1)\times|x| + (4\times T+4) & \text{if } |x|\ mod\ 3 = 1; \\ k < (2\times T-1)\times|x| + (4\times T+3) & \text{if } |x|\ mod\ 3 = 2. \end{cases} \tag{6.6}$$

In conclusion, we pick all strings y such that y shares at least k trigrams by the Equation 6.6 with any given x ($|x| > 3$) and compute the similarity of them only in order to avoid unncessary comparisons. This method is guaranteed not to miss any pair of similar items. The algorithm combined with the approach presented in Section 6.2.1 is summarized in Figure 6.2. The algorithm assumes that the index provides a function, $contain(t)$ for retrieving all strings in S containing a trigram t and $maximum\_trigram$, the maximum number of trigrams found in S. We implemented this function as a hash table by building a string index accessed through trigrams.

Experimental Results

In this subsection, we perform a benchmark test for the proposed optimization schemes on two real data sets collected from the Internet. We

```
Input: $s$ is the given string.
       $S$ is the set of strings, $s_1$, $s_2$, ..., $s_n$.
Output: $S'$ is the set of strings similar to $s$. ($S' \subseteq S$)
Parameter: $T$ is the similarity threshold.

Function RetrieveUsingTrigramIndex ($s$, $S$)

$S' := \emptyset$.
$x := length(s)$.
Switch ($x$ $mod$ 3)
        Case 0: $minimum\_shared\_trigram := (2 \times T - 1) \times |x| + (4 \times T + 2)$.
        Case 1: $minimum\_shared\_trigram := (2 \times T - 1) \times |x| + (4 \times T + 4)$.
        Case 2: $minimum\_shared\_trigram := (2 \times T - 1) \times |x| + (4 \times T + 3)$.
$Candidate := \emptyset$.
For each trigram $t$ of $s$ do
    For each string $c$ of $contain(t)$ do
        If $c \in Candidate$
            $c.counter := c.counter + 1$.
        Else
            $Candidate := Candidate \cup \{c\}$.
            $c.counter := 1$.
For each string $c \in Candidate$ do
    If $c.counter < minimum\_shared\_trigram$
        $Candidate := Candidate$ - $\{c\}$.
For each string $c_i \in Candidate$ do
    $y := length(c)$.
    If LengthFilter($x$,$y$,$T$) = $True$
        $sim$ := similarity($s$,$c$).
        If $sim \geq T$
            $S' := S' \cup \{c\}$.
Return $S'$.
```

Figure 6.2: The optimization algorithm using trigram index

present results showing the efficiency gained by using the optimization methods presented previously for quickly finding similar string-valued items. 3,000 science fiction (SF) book descriptions automatically extracted from the `Amazon.com` online bookstore are used (See Section 3.3.3). Three fields (title, author, subjects) are used in our experiments. The total number of items for this dataset is 7,854 and the average length of the strings is 179. The task is to determine if two strings are similar to each other using a similarity threshold value, T, for every pair of strings where T is set to 0.7. In the experiments presented in this chapter, white spaces contained in strings are considered as a blank character and upper and lower cases are not distinguished.

Figure 6.3 shows the CPU time for the similarity computation step. The "No Optimization" method compares all pairs of items in each field, "String Length" uses a heuristic to eliminate comparisons between strings with very different lengths, and "String Length + Trigram Index" additionally employs the trigram index to retrieve strings with shared trigrams. This experiment was performed on a Linux/i686 PC. For 500 records the optimization reduces running time by 56.36%. As the number of items increase the effectiveness of the optimization decreases but it is still quite effective even for more records. The results demonstrate that with good heuristics and an efficient indexing method, our approach is scalable to larger datasets by reducing the total number of explicit similarity comparisons between pairs of items.

6.2.2 Retrieving k-Nearest Strings

In this subsection, we present how to utilize a trigram index to efficiently retrieve k-nearest-neighbors for short strings. The problem is to find the most similar k strings out of n strings for a given short string. A simple algorithm is presented for increasing the efficiency of information retrieval searches under the edit-distance framework. This optimization algorithm employs knowledge about the lengths of the strings and a branch-and-bound search stategy in order to examine as few strings as possible.

Optimization

To optimize the search for the nearest-neighbor under string edit-distance, we employed two simple optimization techniques. As in Section 6.2.1, we first utilized the information on the string lengths and adopted the branch-and-bound search by terminating the dynamic programming algorithm for computing distances of pairs of strings as soon as the maximum expected

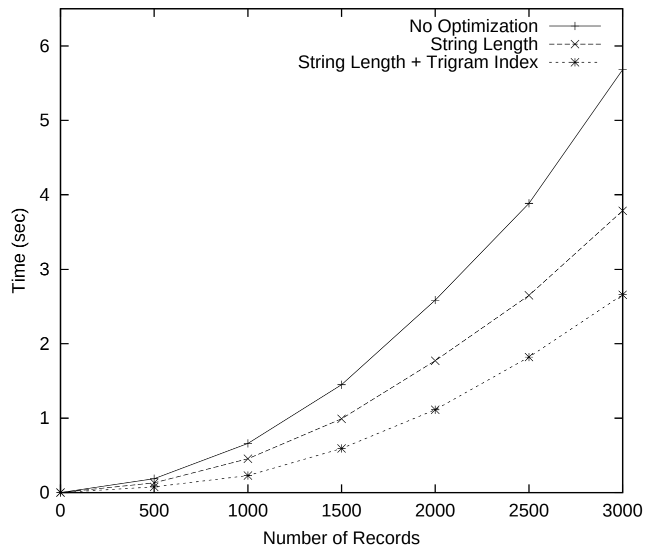


Figure 6.3: Running time for similarity computations

similarity between those strings cannot exceed the current maximum value.

The pseudo-code for this search algorithm is given in Figure 6.4 where $maximum\_distance(x, y)$ returns the maximum distance between two strings with lengths x and y respectively (See Equation 6.1). The subfunction called "SimilarityBranchAndBound" computes the maximum expected distance between two remaining strings after computing the maximum distance at each step of the dynamic programming for affine gap cost function. It then eliminates those strings with low expected similarity since they cannot exceed the current maximum similarity even if the rest of the string is as close as possible to the given string. This algorithm can be easily generalized to k-nearest-neighbor search by maintaining a list of k candidates.

Experimental Results

The same SF book dataset used in the previous experiment (Section 6.2.1) with 3 fields (title, author, subject) are used to evaluate the performance of the proposed optimization. Figure 6.5 shows the CPU time for the nearest-neighbor search. The "No optimization" method stands for linear search where every string is compared with the given string in turn by repeatedly updating the current value of the maximum similarity and "String Length + Bounded Search" employs the proposed optimization scheme. We also

```
Input: $s$ is the given string.
       $S$ is the set of strings, $s_1$, $s_2$, ... $s_n$.
Output: $s'$ is the nearest neighbor of $s$. ($s' \in S$)
Parameter: $T$ is the similarity threshold.

Function FindNearestNeighbor ($s$, $S$)

$x := length(s)$.
$currentNN := s_1$.
$current\_max := 0$.
For each string $s_i \in S$ do
    $y := length(s_i)$.
    $diff := |x - y|$.
    If $diff = 0$
        $minimum\_expected\_distance := 0$.
    Else
        $minimum\_expected\_distance := |x - y| + (gap\_start\_cost - 1)$.
    $maximum\_expected\_similarity := 1 - \frac{minimum\_expected\_distance}{maximum\_distance(x,y)}$.
    If $maximum\_expected\_similarity \geq current\_max$
        $sim := SimilarityBranchAndBound(s, s_i, current\_max)$.
        If $sim > current\_max$
            $current\_max := sim$.
            $currentNN := s_i$.
$s' := currentNN$.
Return $s'$.

SubFunction SimilarityBranchAndBound ($s_1$, $s_2$, $current\_max$)
$s_1 := a_1 a_2 ... a_n$ where $a_i$ is the $i$-th character of $s_1$.
$s_2 := b_1 b_2 ... b_m$ where $b_j$ is the $j$-th character of $s_2$.
For each character $a_i$ in $s_1$ do
    For each character $b_j$ in $s_2$ do
        compute costMaxtrix[i][j] under affine gap function.
    $maximum\_column := max(costMatrix[i][j])$ for all $j := 1$ .. $m$.
        $j' := j$ such that $costMatrix[i][j'] = maximum\_column$.
    $maximum\_expected\_similarity :=$
    $maximum\_expected\_similarity(a_{i+1} a_{i+2} .. a_n, b_{j'+1} b_{j'+2} .. b_m)$.
    If $(maximum\_column + maximum\_expected\_similarity) < current\_max$
        Return 0.
Return $costMatrix[n][m]$.
```

Figure 6.4: The optimized search algorithm

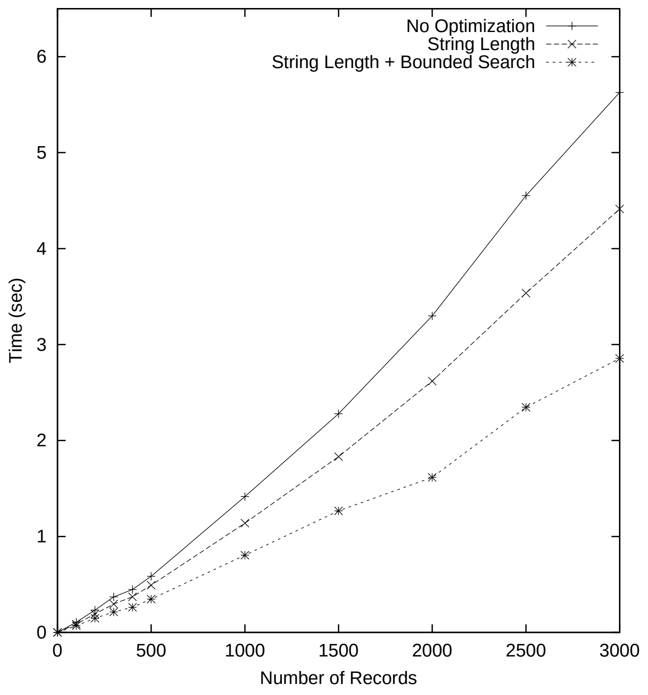


Figure 6.5: Running time for nearest-neighbor search

included the performance results from the version which uses string length information only ("String Length"). For 3,000 records the optimization reduces running time by 49.25%. The results show that our approach with optimization scales up to large datasets.

6.3 Fast Retrieval of Similar Documents

6.3.1 Retrieving Similar Documents Using a Threshold

For longer documents represented as BOWs, a different optimization approach is required. One of the well-known methods developed in the IR community is the inverted index, an index to a set of texts of the words in the texts (Baeza-Yates & Ribeiro-Neto, 1999). Each index entry gives the word and a list of texts in which the word appears. Figure 6.6 summarizes our optimization algorithm.

The optimization technique is based on the observation that calculating exact similarity is not necessary to know that two documents are "simi-

```
Input: $d$ is the given document.
       $D$ is the set of documents, $d_1$, $d_2$, ... $d_n$.
Output: $D'$ is the set of documents similar to $s$. ($D' \subseteq D$)

Function RetrieveUsingInvertedIndex ($d$, $D$)

Let $W$ := set of all words in $d$, $w_1$, $w_2$, ... $w_m$.
Sort $W$ by decreasing idf (inverse document frequency).
$A := \emptyset$.
For each word $w_i \in W$ do
    Let $d_{virtual}$ := document containing all remaining words $(w_{i+1}, w_{i+2} ... w_m)$ in $d$.
    $maximum\_expected\_score := \sum_{i+1}^{m} (score(d(j), d_{virtual}(j)).$
    Read $I_i$, inverted list for $w_i$.
    For each unmarked document $d'$ in $I_i$ do
        If $Current_{d'} \notin Current$ and $Current_{d'}$ is not marked
            $Current_{d'} := 0$.
            $Current := Current \cup \{Current_{d'}\}$.
        $Current_{d'} := Current_{d'} + score(d(i), d'(i))$.
        If normalize($Current_{d'} + maximum\_expected\_score$) $< T$
            mark $Current_{d'}$ and $d'$.
$D' := \emptyset$.
For each unmarked current score $Current_{d'} \in Current$ do
    $sim := normalize(Current_{d'})$.
    If $sim \geq T$
        $D' := D' \cup \{d'\}$.
Return $D'$.
```

Figure 6.6: The optimized search algorithm

lar" using a threshold T ($similarity(d, d') \geq T$). Since $similarity(d, d')$ is computed by accumulating partial scores for each term, we can calculate the maximum expected similarity at each point by assuming the rest of the tokens in one document are exactly the same with those in another and computing the similarity between two virtual documents when the tokens are sorted descendingly by their inverse document frequencies (IDF). The partial score of a term $term_j$ ($score(document_i(term_j), document_{i'}(term_{j'}))$ where $j = j'$) is defined as $idf(term_j)^2 \times count(i, j) \times count(i', j')$ where $count(i, j)$ is the number of occurrences of the term j in document i. If a document cannot be similar to the other even if all the remaining tokens are shared by the two documents, then the algorithm stops and does not compute the exact similarity between them.

Using the inverted index, we are able to address the scaling problem with BOW documents as shown in Figure 6.7. For the "Inverted Index" method, documents that have no shared term with the given document are simply not considered for similarity computation. The "Inverted Index + Optimization" follows the algorithm shown in Figure 6.6. The same set of data used in the previous experiment (Section 6.2.1) was used with the same threshold, $T = 0.7$. BOW-translatable slots such as reviews, comments, synopses and subject were used in this experiment. The total number of items for this dataset is 5,426. The results show that the running time can be greatly reduced even by using a simple indexing technique, employing the inverted index to efficiently retrieve documents with shared terms.

6.3.2 Retrieving k-Nearest Documents

To retrieve k similar textual items when the items are BOWs, we consider optimization techniques suggested earlier in the IR community (Lucarella, 1988). Traditional IR systems have extensively studied the problem of retrieving k-nearest-neighbor documents for a query q, when documents and queries are represented as vectors of terms. We adopted a technique called "partial ranking with document-at-a-time evaluation" (Turtle & Flood, 1995) for effectively obtaining k-nearest neighbors of the given document. In a nutshell, it operates by keeping track of the top k documents as evaluation progresses and terminates evaluation of a document as soon as the maximum score that the document could achive would not place it in the current set of top ranked documents. With this evaluation technique, cost per query is shown to be greatly reduced (Turtle & Flood, 1995).

With the same set of data used in the previous experiment (Section 6.2.2) and $k = 1$, Figure 6.8 shows the CPU time for the similarity computation

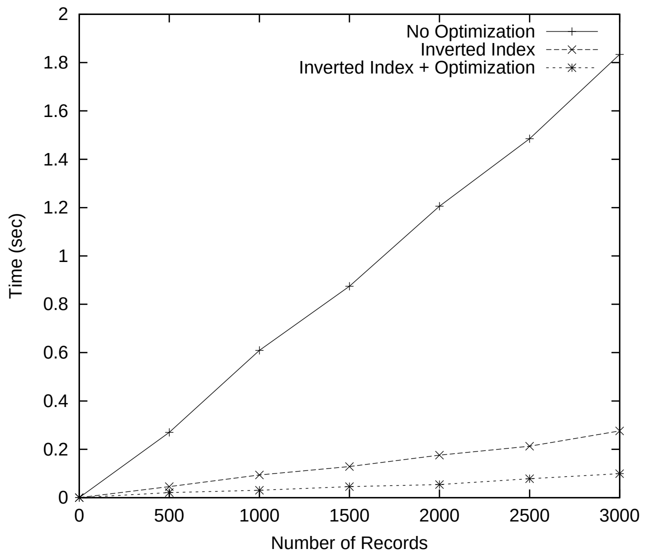


Figure 6.7: Running time for similarity computations

step. The "No Optimization" method compares all pairs of items to find the nearest neighbors and "Optimization" employs the algorithem suggested by Turtle and Flood (1995). The optimization reduces running time by 79.25% on average.

6.4 Summary

The larger the size of available documents becomes, the more critical trade-offs between speed and accuracy emerge, since accurate but slow methods may not be practical in many applications. In this section on the scalability of our approach, we presented methods to efficiently retrieve similar text-valued items in text-mining systems. Straight-forward algorithms, such as those based on nested loops, typically require $O(N^2)$ similarity computations. This quadratic scaling hinders its use when we tackle increasingly larger data sets.

For a given problem of finding *all* similar short-string items under a similarity threshold, we derived an optimized algorithm utilizing a trigram index. For another problem of finding the k most similar short-string items, we presented a simple optimization using information about the length of the given strings for short strings and a branch-and-bound search technique for

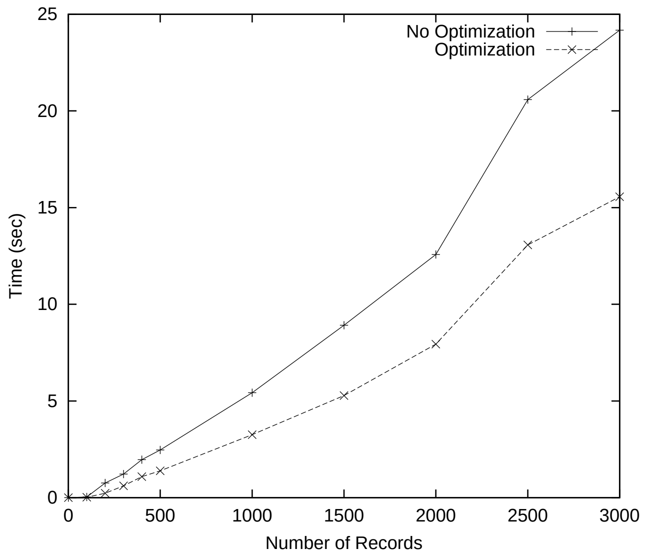


Figure 6.8: Running time for nearest-neighbor search

longer documents. For longer documents, we employed the inverted index, a widely used technique in the information retrieval field. The optimized algorithms are based on the fact that most similarity calculations in the naive algorithm are redundant. Experimental results on a real-life data illustrate that our approach can greatly reduce the total time required for finding similar items. The results demonstrate that with good heuristics and efficient indexing methods, our approach is scalable to larger datasets by reducing the number of explicit similarity comparisons between pairs of items.

Chapter 7

Experimental Comparison of TextRISE and SoftApriori

In previous chapters, we presented two rule mining systems for learning soft-matching rules: TextRISE (Chapter 4) and SoftApriori (Chapter 5). Both of them have their own strengths and weaknesses. We hypothesize that TextRISE which induces soft-matching rules by generalization learns more accurate rules while SoftApriori focuses on efficient mining of soft-matching rules. In this chapter, we present experimental results with TextRISE and SoftApriori on one set of data: book descriptions. The soft-precision and soft-recall are compared to demonstrate the advantage of mining soft-matching rules.

7.1 Experimental Methodology

The experiments presented here are performed on the book data set (Section 3.3.3). To evaluate the performance of the system with varying amounts of training data, we ran tests with smaller subsets of the training examples for each test set and produced learning curves.

We compare four systems in the experiments: TextRISE, SoftApriori, nearest-neighbor, and Apriori. The first two systems discover soft-matching rules while the last one mines hard rules. For these experiments, we used the default values for all parameters of the SoftApriori algorithm as presented in Section 5.3.

To determine the accuracy of a set of rules, we introduce soft-precision and soft-recall with respect to predicting the presence of items in a record from other items in that record. *Soft-precision* is defined as the percentage of predicted items that are (softly) actually present and *soft-recall* is defined as the percentage of actual items that are (softly) correctly predicted. We also report *soft-F-measure* which is the harmonic mean of soft-recall and soft-precision. Soft-F-measure is defined as in Equation 3.3 using soft-precision and soft-recall in place of precision and recall.

The algorithm for computing soft-precision and soft-recall is presented in Figure 7.1. This method is a general extension of the test algorithm shown in Figure 5.3. Instead of adding 1 for each match, this algorithm accumulates the actual similarity between the consequent of rules and the matched part of examples. The notation A' in the algorithm represents the matched part in an example for the antecedent A of a rule.

7.2 Results and Discussion

In order to measure the predictive accuracy of discovered rules, we have performed a ten-fold cross-validation procedure. In each test, the soft-precision and the soft-recall of the system are reported. The results are summarized in Figure 7.2 and and Figure 7.3. Minium support and confidence, and similarity threshold are set to 2%, 10%, 0.7 respectively.

As previously shown in Chapter 4 and Chapter 5, TextRISE performs better than the simple nearest neighbor and soft-matching association rules are more accurate than hard-matching association rules. In addition, the accuracy of the TextRISE rules is consistently highter than others, including soft-matching association rules. Nearest-neighbor provides higher recall, but suffers from lower precision. Training accuracy also shows similar patterns. Differences for each pair of systems were evaluated by a two-tailed, paired t-test to determine if they were statistically significant ($p < 0.05$). Overall, the results show that soft rules are generally better than hard rules and especially TextRISE produces more accurate rules than soft-matching association rules. As discussed in Section 4.3.2, the accuracies are relatively low since predicting textual slots is a hard task.

7.3 Training Time

We measured the training time for TextRISE and SoftApriori to compare the running time of two algorithms. Figure 7.4 shows the evolution of running time with the number of examples for TextRISE and SoftApriori on the book data set. This experiment was performed on a Linux/i686 PC. SoftApriori is much faster than TextRISE. In association rule min-

```
Input: $\mathcal{D}_{test}$ is the test database.
       $Rules$ is the rule set.
Output: $soft\text{–}precision$ and $soft\text{–}recall$ as measured on $\mathcal{D}_{test}$.

Function ComputeAccuracy ($\mathcal{D}_{test}$, $Rules$)

$fired$ := 0.0.
$total$ := 0.0.
$matched$ := 0.0.
$predicted$ := 0.0.
foreach record $R \in \mathcal{D}_{test}$ do
    /* precision */
    foreach $rule$ $(A \Rightarrow c) \in Rules$ do
        if (($rule$ is hard and $A \subseteq R$) or ($rule$ is soft and $A \subseteq_{soft} R$))
            if $rule$ is hard then $A' := A$.
                              else $A' := X$ s.t. $X \subseteq R$ and $X \sim A$.
            $fired := fired + 1$.
            $matched := matched$ + similarity($c$, $c'$)
                         where $c' := \arg\max_{c' \in (R - A')} similarity(c, c')$.
    /* recall */
    foreach $r \in R$ do
        $total := total + 1$.
        if there exists a $rule$ $(A \Rightarrow c) \in Rules$ s.t.
           $c \sim r$ and (($rule$ is hard and $A \subseteq R - \{r\}$)
           or ($rule$ is soft and $A \subseteq_{soft} R - \{r\}$))
           $predicted := predicted$ + similarity($c$, $r$).
$soft\text{–}precision := matched/fired$.
$soft\text{–}recall := predicted/total$.
Return ($soft\text{–}precision$, $soft\text{–}recall$).
```

Figure 7.1: Evaluation algorithm

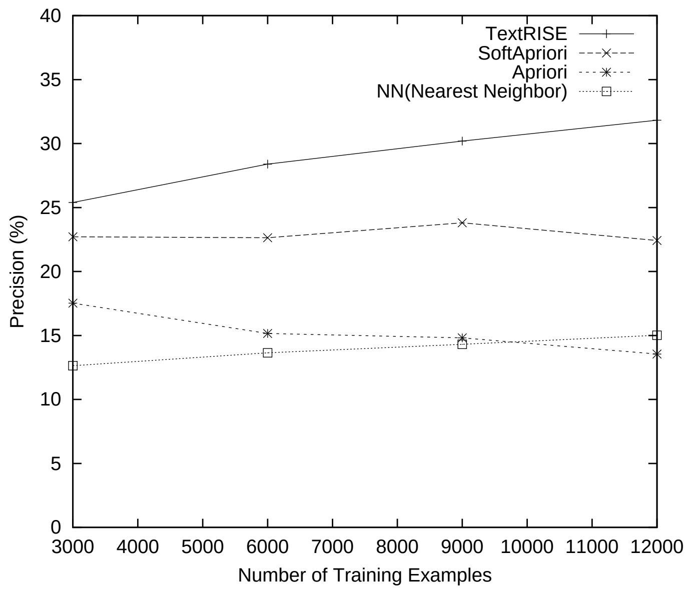


Figure 7.2: Test accuracies: Soft-precision

ing, any association between features is to be discovered, not just ones that predict a particular feature or class. On the other hand, classification rules are learned to predict a specific slot. SoftApriori predicts any attribute, not just one to be predicted as in TextRISE. This graph shows that SoftApriori runs consistently faster than TextRISE as the number of training examples increases.

7.4 Summary

In this chapter, we evaluate the quality of the discovered rules on independent data by measuring the similarity of predicted text and actual text. By comparing results to the predictions made by traditional hard-matching rules and nearest neighbor method, we demonstrate the advantage of mining soft-matching rules. TextRISE induces more accurate prediction rules while SoftApriori discovers soft-matching association rules efficiently. We introduced new measures for evaluating soft-matching rule mining systems: soft-precision and soft-recalls. Overall, soft-matching rules were superior to hard-matching rules in terms of accuracy as hypothesized, thereby indicating that our approach shows considerable promise.

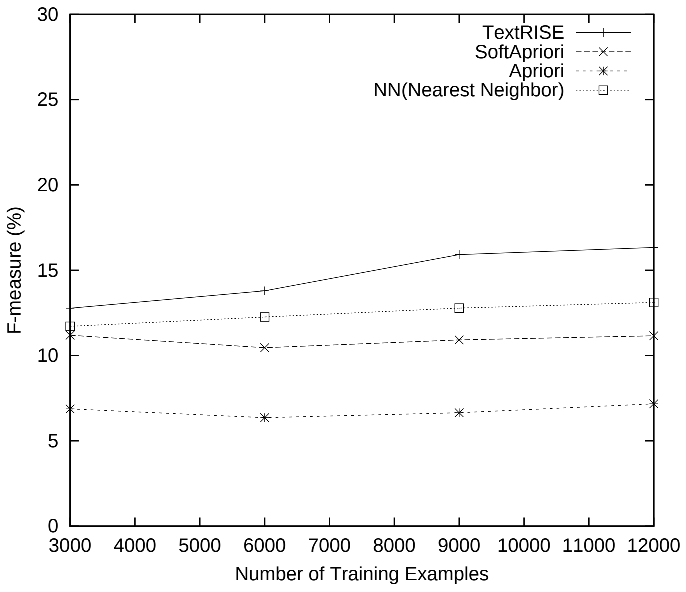


Figure 7.3: Test accuracies: Soft-F-measures

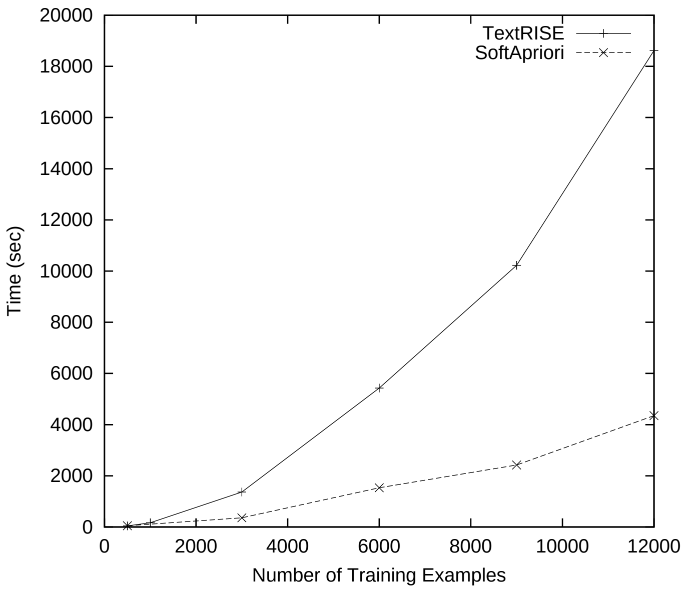


Figure 7.4: Training time

Chapter 8

Using Mined Rules in Improving Information Extraction

In the DiscoTEX framework for integrating IE and KKD, IE benefits KDD by extracting structured data from textual documents, which can then be mined using traditional methods. A less obvious interaction is the benefit that KDD can in turn provide to IE. The predictive relationships between different slot fillers discovered by KDD can provide additional clues about what information should be extracted from a document. This chapter reports experiments in the computer-related job-posting domain demonstrating that predictive rules acquired by applying KDD to an extracted database can be used to improve the accuracy of information extraction.

8.1 Introduction

The general DiscoTEX framework described in Chapter 3 serially combines an information extraction system and a KDD module. Information extraction and data mining can be integrated for the mutual benefit of both tasks. IE enables the application of KDD to unstructured text corpora and KDD can discover predictive rules useful for improving IE performance (Figure 3.1). This chapter explores the mutual benefit that the integration of IE and KDD can provide. DiscoTEX includes a capability for improving the recall of the learned IE system by proposing additional slot fillers based on learned prediction rules as shown in Figure 8.1.

The predictive relationships between different IE slot fillers discovered

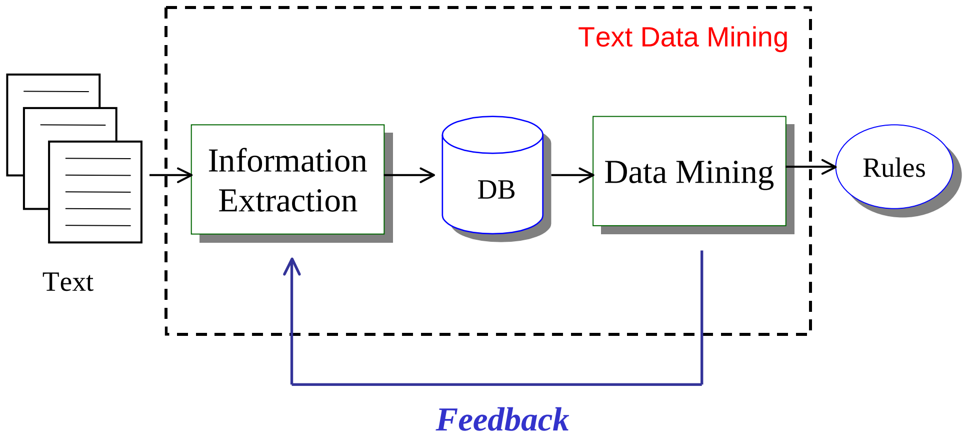


Figure 8.1: Overview of IE-based text mining framework with feedback loop

by KDD can provide additional clues about what information should be extracted from a document. For example, suppose we discover the following rule from data on programming languages and topic areas extracted from a corpus of computer-science job postings: "SQL" $\in$ *language* $\rightarrow$ "Database" $\in$ *area*. If the IE system extracted "SQL" for the *language* slot but failed to extract "Database" for the *area* slot, we may want to assume there was an extraction error and add "Database" to the *area* slot. Since typically the *recall* (percentage of correct slot fillers extracted) of an IE system is significantly lower than its *precision* (percentage of extracted slot fillers which are correct) (DARPA, 1998), such predictive relationships can be productively used to improve recall by suggesting additional information to extract.

8.2 Experiments with Hard-Matching Rules

In this section, we describe the initial system with the feedback loop and report experiments with the initial DiscoTEX (Section 3.4).

8.2.1 The Algorithm

As shown in Section 3.4, we induce rules for predicting the information in each database field given the information in all other fields after constructing an IE system. In order to discover prediction rules, we treat each slot-value pair in the extracted database as a distinct binary feature and learn rules for predicting each feature from all other features. Similar slot fillers are first collapsed into a pre-determined standard term. The experiments in this chapter employ C4.5RULES to induce rules from the resulting binary data by learning decision trees and translating them into pruned rules.

After mining knowledge from extracted data, DiscoTEX uses the discovered rules to predict missing information during subsequent extraction. Tests of IE systems usually consider two performance measures, *precision* and *recall* defined as:

$$precision = \frac{number\_of\_correct\_fillers\_extracted}{number\_of\_fillers\_extracted} \tag{8.1}$$

$$recall = \frac{number\_of\_correct\_fillers\_extracted}{number\_of\_fillers\_in\_correct\_templates} \tag{8.2}$$

Also, F-measure was introduced to combine precision and recall and is computed as Equation 3.3.

Since the set of potential slot fillers is very large and not fixed in advance, and since only a small fraction of possible fillers is present in any given document, these performance metrics are generally more informative than the accuracy of predicting the presence/absence across all slot-value pairs.

Many extraction systems provide relatively high precision, but recall is typically much lower. Previous experiments in the job postings domain showed Rapier's precision (e.g. low 90%'s) is higher than its recall (e.g. mid 60%'s) (Califf, 1998). Currently, Rapier's search focuses on finding high-precision rules and does not include a method for trading-off precision and recall. Although several methods have been developed for allowing a rule learner to trade-off precision and recall (Cohen, 1996a), this typically leaves the overall F-measure unchanged.

By using additional knowledge in the form of prediction rules mined from a larger set of data automatically extracted from additional unannotated text, it may be possible to improve recall without unduly sacrificing precision. For example, suppose we discover the rule `SQL∈language` $\rightarrow$ `Database∈area`. If the IE system extracted `SQL∈language` but failed to extract `Database∈area`, we may want to assume there was an extraction error

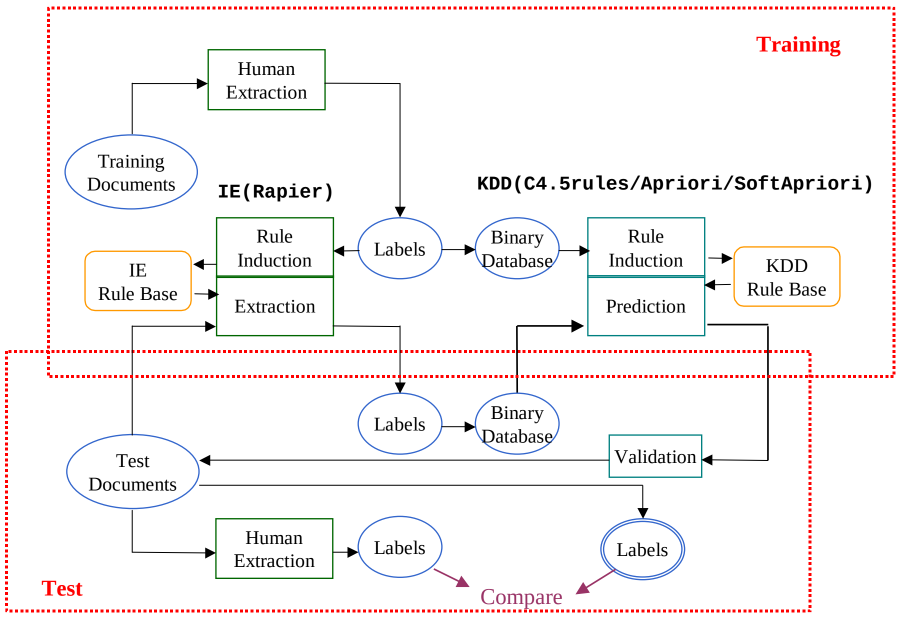


Figure 8.2: The system architecture

and add `Database` to the area slot, potentially improving recall. Therefore, after applying extraction rules to a document, DiscoTEX applies its mined rules to the resulting initial data to predict additional potential extractions. The final decision whether or not to extract a predicted filler is based on whether the filler (or any of its synonyms) occurs in the document as a substring. If the filler is found in the text, the extractor considers its prediction confirmed and extracts the filler. Mined rules that predict the absence of a filler are not used to remove extracted information since there is no analogous confirmation step for improving precision.

The overall architecture of the final system is shown in Figure 8.2. Documents which the user has annotated with extracted information, are used to create a database. The rule miner then processes this database to construct a knowledge base of rules for predicting slot values. These prediction rules are then used during testing to improve the recall of the existing IE system by proposing additional slot fillers whose presence in the document are confirmed before adding them to final extraction template. The last step of validation is made by confirming if the predicted string appears in the document.

8.2.2 Experimental Results

Experimental Methodology

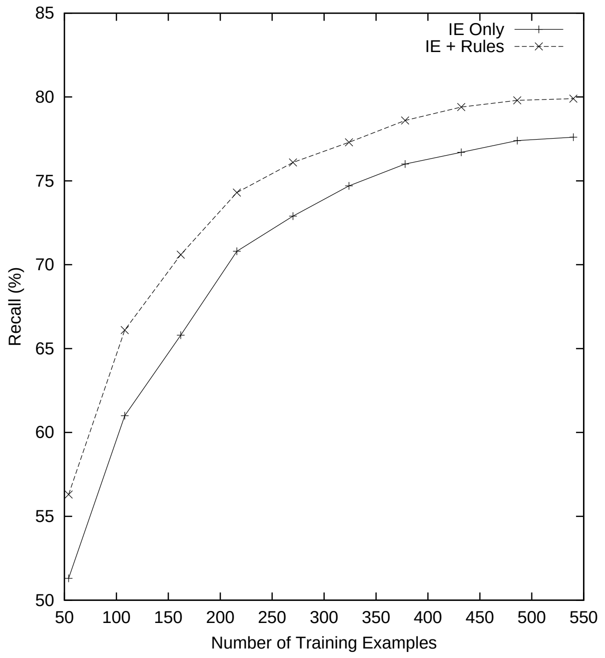
To test the overall system, 600 user-annotated computer-science job postings (Section 3.3.1) were collected. 10-fold cross validation was used to generate training and test sets. In addition, 4,000 unannotated documents were collected as additional optional input to the text miner. Rules were induced for predicting the fillers of the `languages`, `platforms`, `applications`, and `areas` slots, since these are usually filled with multiple discrete-valued fillers and have obvious potential relationships between their values. The `title` slot is also used, but only as a possible antecedent condition of a production rule, not as a consequent. The `title` slot has many possible values and is difficult to predict; however, may be useful as a predictor since fillers such as `Database Administrator` can help determine other values. In this experiment, we use the simpler version of RAPIER that employs only word and part-of-speech constraints since WordNet classes provide no additional advantage in this domain (Califf & Mooney, 1999).

Results and Discussion

In order to clearly illustrate the impact of the amount of training data for both extraction and prediction rule learning, the same set of annotated data was provided to both RAPIER and the rule miner. Figure 8.3 shows a comparison between the performance of RAPIER alone (IE alone) and DISCOTEX (IE + Rules) with mined rules. The results were statistically evaluated by a two-tailed, paired t-test. For each training set size, each pair of systems were compared to determine if their differences in recall and F-measure were statistically significant ($p < 0.05$).

As hypothesized, DISCOTEX provides higher recall, and although it does decrease precision somewhat, overall F-measure is moderately increased. One interesting aspect is that DISCOTEX retains a fixed recall advantage over RAPIER as the size of the training set increases. This is probably due to the fact that the increased amount of data provided to the text miner also continues to improve the quality of the acquired prediction rules. Overall, these results demonstrate the role of data mining in improving the performance of IE.

Table 8.1 shows results on precision, recall and F-measure when additional unlabeled documents are used to construct a larger database prior to mining for prediction rules. In this experiment, unsupervised data which has been processed by the initial IE system (which RAPIER has learned from the



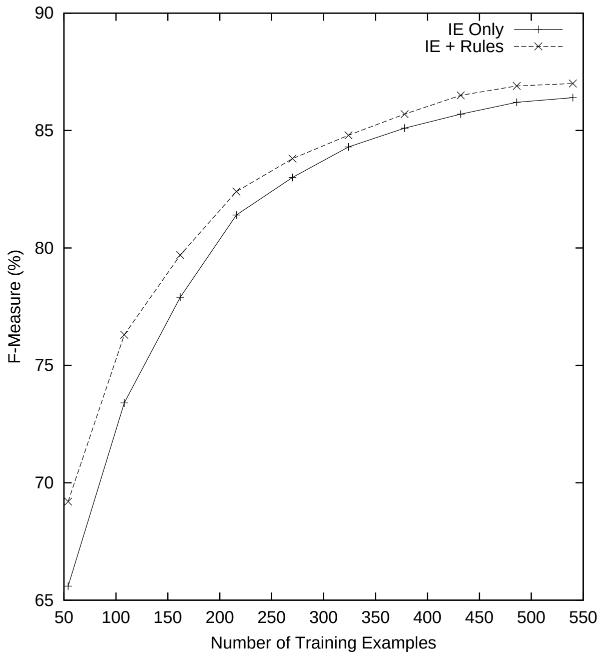


Figure 8.3: Recall and F-measures on job postings for hard-matching rules

supervised data) has been used. The 540 labeled examples used to train the extractor were always provided to the rule miner, while the number of additional unsupervised examples were varied from 0 to 4,000. The results show that the more unsupervised data supplied for building the prediction rule base, the higher the recall and the overall F-measure. Although precision does suffer, the decrease is not as large as the increase in recall.

Although adding information extracted from unlabeled documents to the database may result in a larger database and therefore more good prediction rules, it may also result in noise in the database due to extraction errors and consequently cause some inaccurate prediction rules to be discovered as well. The average F-measure without prediction rules is 86.4%, but it goes up to 88.1% when DiscoTEX is provided with 540 labeled examples and 4,000 unlabeled examples. Unlabeled examples do not show as much power as labeled examples in producing good prediction rules, because only 540 labeled examples boost recall rate and F-measure more than 4,000 unlabeled examples. However, unlabeled examples are still helpful since recall and F-measure do slowly increase as more unlabeled examples are provided.

Number of Examples for Rule Mining	Precision	Recall	F-Measure
0	97.4	77.6	86.4
540 (Labeled)	95.8	80.2	87.3
540 + 1000 (Unlabeled)	94.8	81.5	87.6
540 + 2000 (Unlabeled)	94.5	81.8	87.7
540 + 3000 (Unlabeled)	94.2	82.4	87.9
540 + 4000 (Unlabeled)	93.5	83.3	88.1

Table 8.1: Performance results with unlabeled examples

8.3 Using Soft-Matching Rules to Improve IE

One step in the previous experiments that was performed manually is collapsing similar slot-fillers in the extracted data into a canonical form, e.g. mapping “NT,” “WinNT”, “Windows NT,” and “Microsoft Windows NT” all to a unique term. Although our initial results with this manual step were encouraging, hard-matching rules discovered by standard data mining algorithms may not work for data sets with significant textual variation. We propose mining soft-matching rules instead, which allow non-standardized database entries to match antecedents and consequents based on relevant similarity metrics.

8.3.1 The Algorithm

A benefit of association-rule mining instead of classification-rule induction is that consequents of rules are not predetermined, resulting in efficient mining of all potential associations as part of a single process. When inducing classification rules, a separate learning step must be run for predicting each possible item. For instance, classification rule induction is not as efficient for our job-postings data set with as many as 750 items in 600 documents.

In order to find association rules for extracted data, we first map each extracted filler to an item. A document is represented as a basket of items where each item is a slot-filler pair extracted from the document. By applying SoftApriori to job postings, we mined relationships between items such as “If a computer-related job posting requires knowledge of MFC then it also lists Windows in the list of required skills.”

By forward-chaining on extracted data using soft-matching of antecedents, we can derive additional probable extractions as with hard-matching rules. Pseudocode shown in Figure 8.4 describes the use of mined rules in information extraction.

```
Parameter: $minconf$, $minsup$ - minimum confidence/support.
           $T_{sim}$ - similarity threshold.
           $T_{ex}$ - extraction threshold.
Input: $D_{train}$ - set of labeled documents.
       $D_{test}$ - set of $n$ unlabeled documents.
Output: $L$ - set of $n$ew labels for $D_{test}$.

Function InformationExtraction ($D_{train}$, $D_{test}$)

Build an information extraction rule base, $RB_{IE}$
    (by applying RAPIER to $D_{train}$)
Let $L_{train}$ := set of labeled slot fillers of $D_{train}$.
Build a soft association rule base, $RB$ .
    (by applying SOFTAPRIORI to $L_{train}$
    with parameters $minconf$, $minsup$, and $T_{sim}$)
For each unlabeled document $D_{test}(i)$ do
    Extract slot fillers from $D_{test}(i)$ using $RB_{IE}$.
    Let $L(i)$ := set of extracted slot fillers of $D_{test}(i)$.
    Until no change obtained on $L(i)$ Do
        For each rule $R$ $(X \Rightarrow Y) \in RB$ do
            If $R$ fires on $L(i)$
                For each matching substring $Y'$ in $D_{test}(i)$
                    (with $similarity(Y, Y') \geq T_{sim}$) do
                    $score(Y') := similarity(Y, Y') \times conf(R)$.
                    If $score(Y') \geq T_{ex}$
                        add $Y'$ to $L(i)$.
Let $L$ := ($L$(1),$L$(2),...,$L$($n$)).
Return $L$.
```

Figure 8.4: Algorithm specification for using soft-matching mined rules in IE

The final decision step is modified so that it is based on whether the filler (or any of its "synonyms") occurs in the document. If a string equal or similar to the predicted filler is found in the text, the extractor considers its prediction confirmed and extracts the string. In the previous example, even if the string "Database" is not found in the document, a similar string such as "databases" is still considered for extraction since *similarity*("Database","databases") $\geq T_{sim}$ where T_{sim} is the prespecified threshold for determining a match. The confidence of the rule is also considered in confirming that the rule is strong enough to extract the filler, combined with the similarity information indicating how close the actual string is to the predicted one. In summary, mined soft-matching rules are used during testing to improve the recall of the existing IE system by proposing additional slot fillers whose *similar* strings are confirmed to be present in the document.

8.3.2 Experimental Results

In this section, we demonstrate that using soft-matching rules to predict potential extractions improves the accuracy of IE slightly more than using hard-matching rules. Specifically, we compare the hard-matching rules mined with APRIORI (Agrawal & Srikant, 1994) to soft-matching rules mined with SOFTAPRIORI (Chapter 5) with respect to their ability to improve information extraction from the job postings corpus.

Experimental Methodology

To test the overall system, the same set of computer-science job postings data set (Section 3.3.1) was used. Ten-fold cross validation was used to generate training and test sets for extraction from the set of documents. Rules were mined for predicting the fillers of the `languages`, `platforms`, and `applications` slots, but the `title` slot is not employed.

The similarity threshold, minimum support, and minimum confidence for APRIORI and SOFTAPRIORI were set to 0.70, 5%, and 10%, respectively. Association rules without antecedents (e.g. $\Rightarrow$ `C++`) are also employed. The minium confidence value is set to a low value because the final extraction of a filler is confirmed by checking if the same (hard rules) or similar (soft rules) strings are found in the document or not. Even if some rules make inaccurate predictions, this confirmation step filters out such predictions. The match cost, mismatch cost, gap-start cost, and gap-extend cost parameters for the affine-gap edit distance were set to 0, 3, 3, and 1, respectively. All white spaces in strings are considered as blank characters and upper and lower cases are distinguished only in the IE phase.

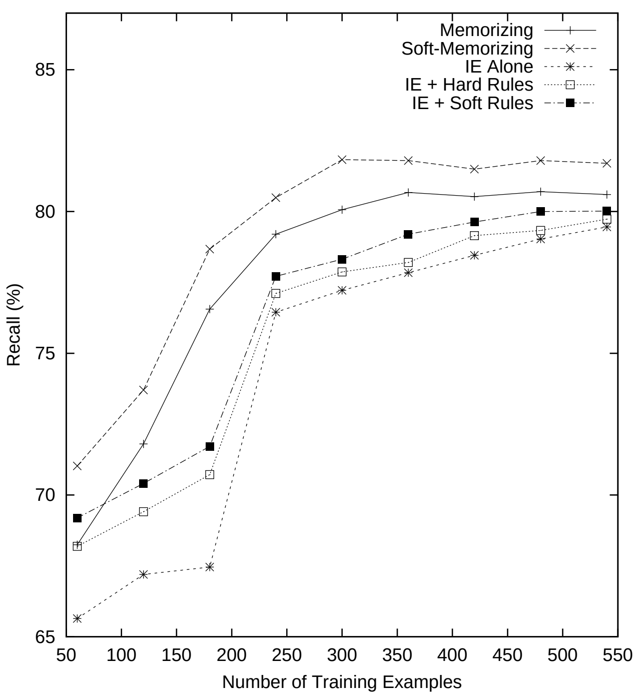
Results and Discussion

To evaluate our system, we compared the performance of RAPIER (Section 2.2) alone, RAPIER aided with hard-matching rules mined by standard APRIORI, and RAPIER with soft-matching association rules mined with SOFTAPRIORI. Figure 8.5 shows the learning curves for recall and F-measure for the job-postings data. The same set of human-annotated training data was provided to both RAPIER and the rule miner as shown in Figure 8.2.

As a benchmark, we also show the performance of a simple baseline (Memorizing) for increasing recall that always extracts substrings that are known fillers for a particular slot. This baseline remembers all slot-fillers that appear at least once in the training data. Whenever a known filler string, e.g. `Java`, is contained in a test document, it is extracted as a filler for the corresponding slot, e.g. `language`. This method has good recall but limited precision since a filler string contained in a document is not necessarily the correct filler for the corresponding slot. For instance, "www" can appear in a document, not in a list of required skills but in a URL of the company's homepage.

We also tested a "soft" version of this baseline (Soft-Memorizing) that extracts all strings that are sufficiently *similar* to known items in the training data. Although this increases recall, it decreases precision even further. For example, "Peoplesoft" remembered as a filler for the `application` slot can cause the system to extract the spurious filler "people". The fact that the F-measure of these baselines are worse than the proposed system demonstrates the additional value of rule mining for improving extraction performance.

RAPIER with soft-matching rules provides higher recall, and in spite of decreasing precision somewhat, overall F-measure is not decreased. For each training set size, systems were compared to determine if their differences in recall were statistically significant using a two-tailed, paired t-test ($p < 0.05$). For all sets of training examples, using soft-matching rules is significantly better than both using hard-matching rules and unaided extraction while using hard-matching rules is also significantly better than unaided extraction. The differences between F-measures were not significant. Although the differences are somewhat small, these results demonstrate the advantage of mining soft-matching rules for improving extraction accuracy without sacrificing F-measures.



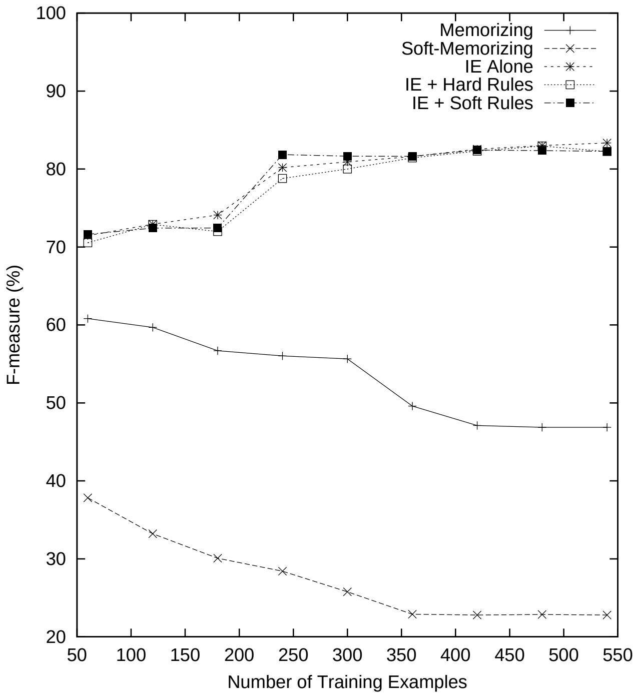


Figure 8.5: Recall and F-measures on job postings

8.4 Summary

In this section, we introduced an approach to using predictive rules mined from extracted data to improve the recall of information extraction. Traditional hard-matching rules could be used for this task in a straightforward way. However, this approach is limited by the requirement that the antecedents and consequents of mined rules exactly match textual items. The normal variation that occurs in textual information frequently prevents such an approach from effectively exploiting many of the potentially-useful predictive relationships in the data. In Chapter 5, we have developed techniques for mining *soft-matching* rules that employ standard text-similarity metrics to discover more subtle relationships in variable textual data. By combining these ideas, we have developed a method for using soft-matching mined rules to further improve the recall of information extraction. Empirical experiments on a real text corpus showed that our new method can more effectively use automatically-discovered knowledge to improve the recall (and F-measure) of information-extraction.

Chapter 9

Related Work

There has been relatively little research exploring the combination of Information Extraction and traditional data mining. Soft-matching used in our rule-learning algorithms have barely been applied to text-mining systems either. In this chapter, we will explain our novelty in using IE for the task of knowledge discovery from text by reviewing earlier work on 1) using existing rule-mining techniques on unstructured or semi-structured text, 2) integrating information extraction and data mining, and 3) handling soft-matching rules for text processing.

9.1 Rule Mining from Text

Besides traditional applications of text processing such as text categorization (Yang, 1999) and text clustering (Manning & Schütze, 1999), discovering rule-based knowledge from unstructured text is an exiting new area for text mining. For example, Knowledge Discovery in Textual Databases (KDT) (Feldman & Dagan, 1995) discovers interesting patterns from text, by establishing a hierarchy of meaningful concepts and looking for mutual connections between the concept nodes. KDT has evolved into the FACT system (Feldman & Hirsh, 1996) with the aid of a well-known data mining technique, *association rule mining*, and DOCUMENT EXPLORER (Feldman, Fresko, Hirsh, Aumann, Liphstat, Schler, & Rajman, 1998) accompanied with an interactive exploration tool. These approaches were applied to Reuters news articles to find interesting relationships between concept items, e.g. natural resources of Latin American countries or business alliances between companies. For example, DOCUMENT EXPLORER discovered rules such as "Chevron Corp and Mobil Corp are likely to be joint venture partners".

Loh, Wives, and de Oliveira (2000) suggested an extension of KDT for web mining by discovering conceptual knowledge from web documents using automated text categorization. In this research, concepts are identified through a text categorization algorithm for the purpose of listing key-concepts and finding correlation between concepts. Ghani et al. (2000) applied several rule induction methods to a database of corporations automatically extracted from the Web as a part of the WebKB project (Craven, DiPasquo, Freitag, McCallum, Mitchell, Nigam, & Slattery, 2000). Data mining techniques used by this system include finding association rules (Apriori), inducing decision trees (C5.0), and learning rules with Foil. Interesting regularities such as "Advertising agencies tend to be located in New York" are discovered from a knowledge base about corporations extracted from the Web. A similar approach has been tested on medical abstracts (Blake & Pratt, 2001).

Lamirel and Toussaint (2000) proposed to extract association rules from a collection of documents by using a variation of SOM (Self-organizing Map) (Kohonen, 1997), but this work has not been extended beyond agglomeration which is essentially document clustering. Recently Pierre (2002) applied the association rule mining algorithm to metadata records generated via automated text categorization in a business domain. Similarly, Ghani and Fano (2002) discover inference rules from a collection of product descriptions by using association-rule mining techniques. For instance, rules such as "If the age group of a consumer is classified as mature, then the trendiness of the products she purchases is low" are discovered from apparel product data.

Two primary aspects distinguish DiscoTEX from other systems that discover rules from text. First, rules learned by these systems are hard-matching rules unlike those produced by TextRISE and SoftApriori. All induced rules must exactly match extracted text, thus the heterogeneity of items in textual databases has not been addressed. The second distinguishing characteristic is DiscoTEX's use of automated information extraction. Most previous systems that mine rules from text do not have an automated process for structuring the documents. For instance, KDT uses texts manually tagged with a limited number of fixed category labels instead of actually using automated text categorization or IE. Similarly, FACT (Feldman & Hirsh, 1996), which finds associations amongst keywords from text documents, does not have an automated routine for labeling the documents with keywords, which can be viewed as the basic level of information extraction. Document Explorer extracts terms to label a document in a more automatic manner, but it is still restricted to highlighting selective terms based on predetermined syntactic patterns such as "noun-noun" or "adjective-noun" (Feldman et al., 1998). One of the limitations for these approaches is that they require a substantial amount of background knowledge provided by a domain expert in advance.

9.2 Integrating IE and Data Mining

There has been relatively little research exploring the integration of IE and KDD. One earlier work related to our approach in spirit can be found in Conrad and Utt (1994). Instead of using information extraction as a preprocessing step for handling natural language texts, they assume structured textual databases as an input and try to find relationships between extracted features.

Etzioni (1996) discusses applying data mining techniques to Web resources available on the Internet. He identifies the significance of using information extraction in building a web mining system with an emphasis on the scalability problem. However, information extraction systems surveyed in this article are very application-oriented and domain-specific, e.g. extraction of answers for frequently asked questions (FAQ) in FAQ-Finder (Hammond, Burke, Martin, & Lytinen, 1995) or extraction of product information from web vendors for a shopping agent, Shopbot (Doorenbos, Etzioni, & Weld, 1997).

KDT (Feldman & Dagan, 1995) and Document Explorer (Feldman et al., 1998) suggest the use of IE in text mining in an indirect way; however, as stated previously, they do not actually use automated text categorization or IE and the paper does not discuss using mined knowledge to improve extraction. In addition, Document Explorer assumes semi-structured documents such as Standard Generalized Markup Language (SGML) text unlike DiscoTEX developed for general natural-language text.

Several natural language processing systems use information extraction. One of those earlier efforts is found in Riloff (1996). It uses a dictionary of extraction patterns, learned originally for IE, to classify text documents. This can be viewed as an indirect use of IE, since it does not employ a full IE system as a component for a larger system but utilizes by-products of an IE learning process.

More recently, IE began to be used for applications such as machine translation (MT) or question-answering (QA). White, Cardie, Han, Kim, Lavoie, Palmer, Rambow, and Yoon (2000) help analysts perform information-filtering tasks on foreign language documents, by making IE techniques based on English transferable to other languages. AutoSlog (Riloff, 1993) is combined with a machine translation system to develop an English information access gateway to newspapers published electronically in foreign countries. QA is another complicated task consisting of understanding questions, locating possible answers from a database, a document corpus, or the Web, and presenting the most reasonable answer. Textract (Srihari & Li, 1999), presented in the QA track of the TREC (Text REtreival Conference)-8 test, answers natural language questions such as "Who won the 2003 Nobel Peace Prize?" by combining a named entity recognizer with other necessary components for QA, e.g. question processors and answer search engines.

Although all of the above research acknowledged the use of information extraction as an essential component for doing other natural-language understanding tasks, none of these concerned an important application of IE, constructing structured textual databases from raw text, for use in text mining.

Cohen (2003a) has recently proposed using feedback from a link analysis module to boost a text-classification system. This system is related to our approach to improve IE with aid of a KDD in spirit since it also tries to boost the underlying learner (web page classifier) by utilizing feedback from a KDD module. Cohen et al. (2002) suggests using a feedback combination of an HTML parser and a higher-level wrapper. A parser for HTML tables and lists used in this research can be viwed as a form of IE since it also transforms (HTML) documents to highly-structured concept hierarchies. However, none of these projects use explicit information extraction in a general way for further utilization of the extracted concepts.

The use of Web-based statistics (search-engine hit counts) to improve the precision of information extraction has been also proposed recently (Soderland, Etzioni, Shaked, & Weld, 2004). Instead of increasing recalls by additionally extract fillers, this system based on the KnowItAll information extraction module (Etzioni et al., 2004) attempts to increase precision by filtering out extracted fillers using Web statistics.

McCallum and Jensen (2003) proposed a probabilistic framework for unifying information extraction and data mining. A general approach for using statistical relational models to integrate IE and KDD is presented, but an actual implementation and experimental results for this approach are still forthcoming. In this work, data mining run on a partially-filled database finds patterns that provide "top-down" constraints to information extraction. On the other hand, information extraction provides a set of "bottom-up" hypotheses to data mining that can handle uncertainty information. However, unlike our approach which uses collective knowledge mined from an entire set of documents, this framework focuses on interaction between IE and KDD within a document.

9.3 Mining Soft-Matching Rules

Traditionally, the "bag-of-words" model (Baeza-Yates & Ribeiro-Neto, 1999) in Information Retrieval (IR) has been widely used to handle texts. However, unlike simple tasks such as document matching, ranking, and clustering, soft-matching has not been adequately addressed in rule-mining tasks.

Recently, Cui, Kan, and Chua (2004) proposed an unsupervised learning system that induces soft-matching patterns for classifying sentences in online news articles. To accommodate the diversity of sentence structure, flexible, soft patterns are introduced and employed. Soft patterns include not only lexical tokens but also part-of-speech (POS) tags and puctuation.

Whirl is a query processing system that combines traditional database and IR methods by introducing a "soft join" operation (Cohen, 1998). In Whirl, all information is assumed to be represented in a relational model in which every element of every tuple contains free text. Although Whirl and DiscoTEX share a focus on soft-matching rules for text processing, rules in Whirl must be written by the user while DiscoTEX tries to discover such rules automatically.

Compared to automated data cleaning or duplicate detecting methods that impose a single normalization on the data items (Cohen et al., 2000; Hernández & Stolfo, 1995; McCallum et al., 2000b; Monge & Elkan, 1996; Winkler, 1999), mining soft-matching rules dynamically clusters data items into different groups depending on the association under consideration, i.e. each discovered rule may group items into different similarity-based equivalence classes. For example, "Windows NT" must be placed either in the "NT" or "Windows" group in the normalization approach, while our algorithm allows it to belong to both clusters, depending on the inference that is being made.

Chapter 10

Future Work

We will address a number of issues in future research in this chapter. These fall into two primary areas. First there are several enhancements which can be made to the DiscoTEX system: using background information, mining more expressive rules, enhacing DiscoTEX, and improving information extraction. Next, we intend to explore the applicability of our framework to other text mining tasks. Each of these areas of future research is discussed in some detail below.

10.1 Using Background Information

One shortcoming of DiscoTEX is that it does not consider the use of prior information or metadata. Incorporating domain knowledge has been one of the important topics in machine learning and data mining (Witten & Frank, 1999). Metadata often involves relations among attributes such as semantic relation, causal relation, or functional dependencies. As a result of the explotion of the amount of electronically-available data, it is often the case that other sources of knowledge are easily accessible and exploitable. A potential extension of DiscoTEX is to use the WordNet (Fellbaum, 1998) hierarchy of *hypernyms* to generate generalizations that takes semantics into account. WordNet hypernyms have been shown to be helpful in improving text categorization (Scott & Matwin, 1998). For example, the two rules:

- thermodynamics $\in$ *subject* $\rightarrow$ heat, modern, theory, waves $\in$ *synopses*
- optics $\in$ *subject* $\rightarrow$ electromagnetics, laser, waves $\in$ *synopses*

might be minimally generalized to the rule

- physics $\in$ *subject* $\rightarrow$ waves $\in$ *synopses*

if a semantic lexicon provided by WordNet is utilized. "Thermodynamics" and "optics" have the common parent, "physics", in the hypernym tree. The current implementation of DiscoTEX generates an empty filler slot for the `subject` slot in this case because it is not able to recognize the semantic relationships between the two slot fillers. This change requires a redefinition of distance between words in terms of the WordNet hierarchy as in Basu, Mooney, Pasupuleti, and Ghosh (2001). Once the distances are redefined, a generalized association rule mining algorithm (Srikant & Agrawal, 1995) for learning abstract rules can be applied.

Like in Zelikovits and Hirsh (2002) or Pereira, Tishby, and Lee (1993) which use additional "background text" or semantic information given by LSI (Latent Semantic Indexing) (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990) in text classification or word clustering, utilizing statistical measures of semantic similarity might be a another option. A similar approach has been proposed by Choudhary and Bhattacharyya (2002) for text clustering. In this work, a document clustering algorithm based on UNL (Universal Networking Language) (Uchida, Zhu, & Della, 2000), a semantic representation for sentences, is developed. The OpenMind commonsense database, the Cyc knowledge base (Lenat, 1997), or multilingual ontologies/dictionaries such as BRICO (Haase, 2000) are potential sources for background knowledge.

We may also use domain-specific sources of semantic information such as dictionaries of programming langauges, job titles, companies, places, etc. Henze and Nejdl (2002) developed an ontology for the programming language Java and Hotho, Staab, and Stumme (2003) utilize it in clustering eLearning course Web pages about Java. For instance, it would be interesting to attempt to have a semantic class consisting of programming languages which can be expanded to incorporate new languages.

Incoporating structural information is a viable option for utilizing background knowledge. There is a growing need to handle semi-structured documents written in mark-up languages. The current version of DiscoTEX does not have any special ability to cope with Web pages written in HTML or XML (EXtensible Markup Language) (Bray, Paoli, Sperberg-McQueen, & Maler, 2000). Automatically generated web pages in `Amazon` and `IMDb` contain HTML tags, but they are only used by a wrapper. Structural hints given by such tags could be utilized in rule mining as well.

One option for this task is to take the XML document structure tree generated by an XML parser as the representation for a document. Although the problem of handling hierarchical structure has received relatively little attention in the machine learning literature, XML is increasingly being used for exchanging information from data sources. In this case, the generalization of two sets of bags or strings in the current system is replaced by the generalization of two trees of bags or strings. The similarity measures should also be redefined appropriately for this representation. Tree edit distance (Zhang & Shasha, 1989) could be adopted in redefining the distances.

One of the barriers in utilizing the structural information hidden inside the documents is that most current pages on the Web are still in HTML although XML pages are more powerful than HTML ones for describing the structured contents of a page. A method for automatically recognizing the structures in HTML pages is needed in that case (Cohen, 2003b). In this case, tree mining alogorithms for finding frequent subtrees such as TREEMINER (Zaki, 2002) could be applied instead of the general association rule mining algorithms for discovering frequent terms or bags in the current implementation of DISCOTEX. Changes to the system should be relatively straightforward. We believe that text-mining algorithms can be improved by combining content-based information with structural cues if the structure underlying textual data can be recovered.

10.2 Mining More Expressive Rules

Instead of taking individual words, n-grams could be used as additional terms for the vector representation. When applying DISCOTEX to book data, we found that it recognizes "Juvenile Fiction" and "Science Fiction" in the `subject` slot as a BOW with "juvenile(1)", "science(1)" and "fiction(2)", thus missing useful information which could have been maintained if they had been treated as a single term. For example, we could construct a bag-of-uni/bigrams with "juvenile(1)", "science(1)", "fiction(2)", "juvenile-fiction(1)", and "science-fiction(1)" from the same slot-filler.

A related issue is to combine words for a named entity, such as an author's name or a company name. "Stephen" or "King" by themselves in "Stephen King" might not be very useful for mining interesting rules about this author, but "stephen-king" as a whole makes more sense to users looking at this domain. Named entity recognition task, introduced in the MUC-6 (DARPA, 1995), was found to be a relatively easy task because the best system submitted to the competition scored 96.4% in F-measure (business news domain). Automatic extraction of named entities with RAPIER or other machine learning techniques such as (Bikel, Schwartz, & Weischedel,

1999) or (Kelin, Smarr, Nguyen, & Manning, 2003) could be considered for a preprocessing step of building a training set with bags-of-terms.

Another option is to replace the current generalization scheme in TextRISE. For example, we we take the intersection of two BOWs for generalization currently. First, we could generalize to a k-nearest-neighbor method that uses the k closest rules or examples rather than just the single nearest one. The predictions of these k rules could be combined by taking the generalization of the slots (e.g. the average of the BOW vectors) in their consequents. Likewise during learning, rules could be generalized to the k nearest uncovered examples using a similar averaging technique, possibly rounding values to maintain integer counts and simplifying the resulting rules.

We also believe an algorithm for mining sequential association rules (Srikant & Agrawal, 1996) could be applied to textual data without much modification. Sequential rule mining algorithms can discover a rule, "Customers usually purchase wireless cards and routers *after* they bought laptops." from a customer transaction database. While speech and biological sequences have been considered as typical examples for sequential data, sequential data mining algorithms have not been widely used in text mining. Although Ahonen-Myka, Heinonen, Klemettinen, and Verkamo (1999) suggested a similar approach based on the episode rules, adopting sequential association rules has the advantage that they are a straightforward generalization of generic association rules.

10.3 Enhancing DiscoTEX

One of the important issues is the interestingness of the mined rules. Even though we currently rank rules before presenting them to users, this task bears further investigation. The interestingness measures we used currently, confidence and support, are common measures in the KDD community. However, the quality or goodness of rules specific for textual databases could be defined in different ways. For instance, trivial rules such as "If the `title` of a book contains "chemistry", both the `synopses` and the `subject` have "chemistry", too." are very often found in the rule set generated by DiscoTEX. This kind of rule can be left out via simple identity checking filters.

Another idea is to use a semantic network like WordNet (Fellbaum, 1998) again to measure the semantic distance between the words in the antecedent and the consequent of a rule, preferring more "surprising" rules where this distance is larger. For example, this would allow ranking the rule "beer

→ diapers" above "beer → pretzels" since beer and pretzels are both food products and therefore closer in WordNet. Although WordNet provides rich information for a given word such as synonyms, antonyms, hypernyms, and meronyms, the interestingness measure might mostly concern synonyms and hypernyms. We could define the semantic distance between two words as the length of the path between those in the WordNet hierarchy (Basu et al., 2001).

Domain-specific dictionaries constructed in the process of building word vectors for document collections can be used to eliminate uninteresting terms before learning. For instance, in the `synopses` slot of book data, many examples contain "table", and "contents", since a "table of contents" is usually included in the synopses of a book. The rules learned by DiscoTEX therefore contain both "table" and "contents" in their `synopses` slots in many cases, although they are neither informative nor interesting. "Copyright" in the `reviews` slot or any words related to books such as "book" or "read" are other typical examples for this problem. They do not provide any clue to the specific book they describe. Although we already used the TFIDF weighting scheme to prefer words that are dominant in a particular document, eliminating such domain-specific frequent words spread throughout a collection of documents could help with finding more interesting rules.

Using flexible and learnable metrics instead of fixed-cost similarity metrics (Bilenko & Mooney, 2003) for soft matching might be a good option. In this work, a duplicate detection system uses trainable measures, instead of relying on generic and manually-tuned distance metrics, for estimating the similarity of textual items. Although this framework focuses more on duplicate detection in textual databases, a learnable similiary metric for either each database field or database record could be applied to our system. Since DiscoTEX was designed to plug in any similarity metric for each field, adaptively-tuned similarity metrics can be used in place of other metrics. A related issue is comparing our approach of mining soft-matching rules with that of Marlin (Bilenko & Mooney, 2003) in which similar items are first collapsed before traditional hard-matching rules are mined.

Specifically for SoftApriori, an extension considering the actual similarity between items could be explored. The limitation of the current definitions for soft-support and soft-confidence is that they do not reflect the different original support values of individual items nor different degrees of similarities between items. One possible solution to this problem is to redefine the similarity matrix as $similarity(i,j)$ instead of the binary value, $similar(i,j)$. In other words, the similarity matrix is not binary but should be filled with the actual value for similar pairs of items. The optimization method described in Chapter 6 should be redesigned in this case. Using predictive assocation rules which were shown to be more accurate than plain association rules is another option to improve SoftApriori (Bayardo Jr., 1997; Liu, Hsu, & Ma, 1998; Yin & Han, 2003).

10.4 Improving Information Extraction

Currently, we only consider improving the recall of IE. Methods for using mined knowledge to improve extraction precision are also needed. Simply eliminating extracted fillers that are not predicted is too coarse and would likely severely damage recall. One possible solution is to use a variation of *negative* association rules (Savasere, Omiecinski, & Navathe, 1998; Wu, Zhang, & Zhang, 2002). By confidently predicting the *absence* of certain slot values given other extracted information, both precision and recall could potentially be improved.

One great potential impact would be on the utility of the World Wide Web since the Web is an immense, multilingual, freely available corpus. Increasing precision using Web statistics such as PMI (pointwise mutual information) scores (Soderland et al., 2004) could be also considered. The PMI score of a descriminator pattern D and an extraction pattern E are defined as:

$$PMI(D, E) = \frac{Hits(D + E)}{Hits(E)}$$

while $D + E$ is the descriminator pattern with the extraction substituted for the instance. For example, $Hits$('City of <City>', 'Austin') is the number of hits in a search engine with a query "City of Austin". Soderland et al. (2004) showed that the precision of an underlying information extraction system can be improved by providing PMI scores to a KDD module (naïve Bayesian classifier (Mitchell, 1997)) as input. Although we applied the same method on the job-postings and the resumé domain (e.g. computing PMI('programming language', 'java') or PMI('platform', 'natural') with the `AltaVista` search engine[1]), we were not able to improve the information-extraction system substantially. We believe that this is because the Web has too many general terms to descriminate computer-related terms. Using domain-specific ontologies as background knowledge as discussed in Section 10.1 or narrowing the search domain by pre-selecting Web pages dedicated to the topic could solve this problem.

[1]`http://www.altavista.com/`

10.5 Extension to Other Text Mining Tasks

We also believe that additional text mining tasks can be found for which our framework may prove useful. Since IE can be useful for many kinds of text processing, using extracted information for other text mining tasks might be an interesting issue. However, little research has been done in this field. We present here possible extensions of our framework to one of the other text mining tasks: topic detection.

Topic Detection and Tracking (TDT), or novelty detection, is a variant of traditional document classification that allows new classes over a time-series text corpora. Unlike query-based retrieval task or simple categorization task, users do not know in advance what they want to retrieve or what categories there are. Information retrieval and machine learning techniques have been used to identify new events from streams of articles, concentrating on news stories (Yang, Carbonell, Brown, Pierce, Archibald, & Liu, 1999).

As in TopCat (Clifton & Cooley, 1999), we consider a TDT task in a data mining context: preprocessing a collection of articles for identifying key concepts in individual documents, and applying data mining techniques such as frequent item set generation and clustering. Frequent item sets, defined as all item sets that often occur together, identify correlated topics while clustering tracks natural boundaries for neighboring topic periods to detect changes of topic items over time. TopCat uses a named entity recognizer which is limited to extracting information related to "who?" and "where?" questions (e.g. `location`, `organization`, and `person`).

Similar approaches for discovering trends in text databases have been proposed and applied to a database of U.S. patents (Lent, Agrawal, & Srikant, 1997) or Spanish newspapers (y Gómez, Gelbukh, & López-López, 2001), but were limited to the simple framework of mapping each word or phrase to an item. By identifying and highlighting keywords or named entities such as persons, organization, and locations, more interesting trends and rules could be discovered.

Chapter 11

Conclusion

With the dramatic increase in online information in recent years, text mining at the intersection of data mining, natural-language processing, machine learning, and information retrieval, is starting to gain increasing interest. In this dissertaion, we present a new framework for text mining, called DiscoTEX (Discovery from Text EXtraction), which uses a learned information extraction system to transform text into more structured data which is then mined for interesting relationships.

The ability to extract relationships and rules from natural-language texts is an important task with a growing number of potential applications. Information Extraction (IE) is a form of shallow text understanding that locates specific pieces of data from a corpora of natural-language texts. Data Mining or Knowledge Discovery from Databases (KDD) considers the application of statistical and machine-learning methods to discover novel relationships in large relational databases. However, there has been little if any research exploring the interaction between these two important techniques to perform text-mining tasks.

The goal of text mining is to discover knowledge in unstructured text. The related task of IE concerns transforming unstructured text into a structured database by locating desired pieces of information. Although handmade IE systems have existed for a while, automatic construction of information extraction systems using machine learning is more recent. DiscoTEX combines IE and standard data mining methods to perform text mining as well as improve the performance of the underlying IE system. It discovers prediction rules from natural-language corpora, and these rules are used to predict additional information to extract from future documents, thereby improving the recall of IE.

Existing methods for mining rules from text use a hard, logical criteria for matching rules. However, for most text processing problems, a form of soft matching that utilizes word-frequency information typically gives superior results. Therefore, the induction of soft-matching rules from text is an important, under-studied problem. The standard rule mining algorithms have problems when the same extracted entity or feature is represented by similar but not identical strings in different documents. Consequently, we developed an alternate rule induction system for DiscoTex called, TextRISE, that allows for partial matching of textual features. SoftApriori, another rule mining system which is an extension of the general association rule miner, has also been developed.

We presented experimental results applying the TextRISE rule learner and the SoftApriori rule miner to corpora of Internet documents retrieved from the World Wide Web (WWW). The empirical results obtained with two systems on the `Amazon.com` book descriptions data set show that TextRISE focuses on inducing accurate rules by gradually generalizing textual instances while SoftApriori concerns efficient mining of soft-matching rules.

In conclusion, we have presented a general framework for text mining by combining existing IE and KDD technologies. Two rule-learning approaches that use a flexible mechanism both in their rule-learning algorithm and in their classification schemes are developed: TextRISE and SoftApriori. The former was applied to the task of learning inductive rules, which produced better accuracy than the nearest-neighbor approach. The latter was applied to mining association rules for capturing additional relationships, which was shown to give better efficiency in mining soft patterns. Instead of considering the documents as a simple bag of words or a string, we used a flexible method of plugging in a similarity metric for each field. Both rule-learning systems for automated discovery of knowledge from unstructured text were demonstrated to perform better than previous methods in several domains.

Acknowledgements

The research was supported by the National Science Foundation under grant IRI-9704943 and IIS-0117308.

Bibliography

Agrawal, R., Imielinsky, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD-93)*, pp. 207–216.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB-94)*, pp. 487–499 Santiago, Chile.

Aha, D. W., Kibler, D. F., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, *6*(1), 37–66.

Ahonen, H., Heinonen, O., Klemettinen, M., & Verkamo, A. I. (1998). Applying data mining techniques for descriptive phrase extraction in digital document collections. In *Proceedings of the IEEE Forum on Research and Technology Advances in Digital Libraries*, pp. 2–11 Santa Barbara, CA.

Ahonen-Myka, H., Heinonen, O., Klemettinen, M., & Verkamo, A. I. (1999). Finding co-occurring text phrases by combining sequence and frequent set discovery. In Feldman, R. (Ed.), *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99) Workshop on Text Mining: Foundations, Techniques and Applications*, pp. 1–9 Stockholm, Sweden.

Angell, R. C., Freund, G. E., & Willet, P. (1983). Automatic spelling correction using a trigram similarity measure. *Information Processing and Management*, *19*(4), 255–261.

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern Information Retrieval.* ACM Press, New York.

Basu, S., Mooney, R. J., Pasupuleti, K. V., & Ghosh, J. (2001). Evaluating the novelty of text-mined rules using lexical knowledge. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)*, pp. 233–239 San Francisco, CA.

Bayardo Jr., R. J. (1997). Brute-force mining of high-confidence classification rules. In Heckerman, D., Mannila, H., & Pregibon, D. (Eds.), *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pp. 123–126 Newport Beach, CA. AAAI Press.

Bayardo Jr., R. J., & Agrawal, R. (1999). Mining the most interesting rules. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining (KDD-99)*, pp. 145–154 San Diego, CA.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web.. Scientific American.

Berry, M. W. (Ed.). (2003a). *Proceedings of the Third SIAM International Conference on Data Mining(SDM-2003) Workshop on Text Mining*, San Francisco, CA.

Berry, M. W. (2003b). *Survey of Text Mining: Clustering, Classifcation, and Retrieval*. Springer Verlag, Berlin, Germany.

Bikel, D. M., Schwartz, R., & Weischedel, R. M. (1999). An algorithm that learns what's in a name. *Machine Learning*, *34*, 211–232.

Bilenko, M., & Mooney, R. J. (2003). Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)*, pp. 39–48 Washington, DC.

Blake, C., & Pratt, W. (2001). Better rules, fewer features: A semantic approach to selecting features from text. In Cercone, N., Lin, T. Y., & Wu, X. (Eds.), *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM-2001)*, pp. 59–66 San Jose, CA. IEEE Computer Society.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., & Maler, E. (2000). EXtensible Markup Language (XML) 1.0 (second edition). http://www.w3.org/TR/2000/REC-xml-20001006.

Brill, E. (1994). Some advances in rule-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 722–727 Seattle, WA.

Bunescu, R., Ge, R., Kate, R. J., Marcotte, E. M., Mooney, R. J., Ramani, A. K., & Wong, Y. W. (2004). Comparative experiments on learning information extractors for proteins and their interactions. *Special Issue in the Journal Artificial Intelligence in Medicine on Summarization and Information Extraction from Medical Documents*, *31*. To appear.

Califf, M. E. (1998). *Relational Learning Techniques for Natural Language Information Extraction*. Ph.D. thesis, Department of Computer Sciences, University of Texas, Austin, TX. Also appears as Artificial Intelligence Laboratory Technical Report AI 98-276.

Califf, M. E. (Ed.). (1999). *Papers from the AAAI-1999 Workshop on Machine Learning for Information Extraction*, Orlando, FL. AAAI Press.

Califf, M. E., & Mooney, R. J. (1999). Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 328–334 Orlando, FL.

Chakrabarti, S. (2002). *Mining the Web: Analysis of Hypertext and Semi Structured Data*. Morgan Kaufmann, San Francisco, CA.

Chiang, R., Laender, A., & Lim, E.-P. (Eds.). (2003). *Proceedings of the Fifth International Workshop on Web Information and Data Management(WIDM-2003)*, New Orleans, LA. ACM.

Choudhary, B., & Bhattacharyya, P. (2002). Text clustering using semantics. In *Proceedings of the Eleventh International World Wide Web Conference (WWW-2002)* Honolulu, HI. ACM. Short paper.

Ciravegna, F., Basili, R., & Gaizauskas, R. (Eds.). (2000). *Proceedings of the 14th European Conference on Artificial Intelligence(ECAI-2000) Workshop on Machine Learning for Information Extraction*, Berlin, Germany.

Ciravegna, F., & Kushmerick, N. (Eds.), text mining (2003). *Papers from the 14th European Conference on Machine Learning(ECML-2003) and the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases(PKDD-2003) Workshop on Adaptive Text Extraction and Mining*, Cavtat-Dubrovnik, Croatia.

Clifton, C., & Cooley, R. (1999). TopCat: Data mining for topic identification in a text corpus. In Zytkow, J. M., & Rauch, J. (Eds.), *Proceedings of Third European Conference of Principles and Practice of Knowledge Discovery in Databases(PKDD-99)*, pp. 174–183 Prague, Czech Replublic. Springer Verlag. Lecture Notes in Computer Science Vol. 1704.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pp. 115–123 San Francisco, CA.

Cohen, W. W. (1996a). Learning to classify English text with ILP methods. In De Raedt, L. (Ed.), *Advances in Inductive Logic Programming*, pp. 124–143. IOS Press, Amsterdam.

Cohen, W. W. (1996b). Learning trees and rules with set-valued features. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pp. 709–716 Portland, OR.

Cohen, W. W. (1998). Providing database-like access to the web using queries based on textual similarity. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD-98)*, pp. 558–560 Seattle, WA.

Cohen, W. W. (2003a). Improving a page classifier with anchor extraction and link analysis. In Becker, S., Thrun, S., & Obermayer, K. (Eds.), *Advances in Neural Information Processing Systems 15*, pp. 1481–1488 Cambridge, MA. MIT Press.

Cohen, W. W. (2003b). Learning and discovering structure in Web pages. *IEEE Data Engineering*, *26*(3), 3–10.

Cohen, W. W., Hurst, M., & Jensen, L. S. (2002). A flexible learning system for wrapping tables and lists in HTML documents. In *Proceedings of the Eleventh International World Wide Web Conference (WWW-2002)*, pp. 232–241 Honolulu, HI. ACM.

Cohen, W. W., Kautz, H., & McAllester, D. (2000). Hardening soft information sources. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000)* Boston, MA.

Conrad, J. G., & Utt, M. H. (1994). A system for discovering relationships by feature extraction from text databases. In Croft, W. B., &

van Rijsbergen, C. J. (Eds.), *Proceedings of Seventeenth International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-94)*, pp. 260–270 Dublin, Ireland. ACM / Springer.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., & Slattery, S. (2000). Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*, *118*(1-2), 69–113.

Cui, H., Kan, M.-Y., & Chua, T.-S. (2004). Unsupervised learning of soft patterns for generating definitions from online news. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW-2004)* New York, NY. ACM.

DARPA (Ed.). (1995). *Proceedings of the Sixth Message Understanding Evaluation and Conference (MUC-95)*, San Mateo, CA. Morgan Kaufmann.

DARPA (Ed.). (1998). *Proceedings of the Seventh Message Understanding Evaluation and Conference (MUC-98)*, Fairfax, VA. Morgan Kaufmann.

Dave, K., Lawrence, S., & Pennock, D. M. (2003). Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the Twelfth International World Wide Web Conference (WWW-2003)* Budapest, Hungary. ACM.

Deerwester, S. C., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, *41*, 391–407.

Domingos, P. (1996). Unifying instance-based and rule-based induction. *Machine Learning*, *24*, 141–168.

Doorenbos, R. B., Etzioni, O., & Weld, D. S. (1997). A scalable comparison-shopping agent for the World-Wide Web. In *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*, pp. 39–48 Marina del Rey, CA.

Eirinaki, M., & Vazirgiannis, M. (2003). Web mining for web personalization. *ACM Transactions on Internet Technology*, *3*(1), 1–27.

Etzioni, O. (1996). The World-Wide Web: Quagmire or gold mine?. *Communications of the Association for Computing Machinery*, *39*(11), 65–68.

Etzioni, O., Cafarella, M., Downey, D., Kok, S., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D. S., & Yates, A. (2004). Web-scale information extraction in KnowItAll. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW-2004)* New York, NY. ACM.

Feldman, R. (Ed.). (1999). *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99) Workshop on Text Mining: Foundations, Techniques and Applications*, Stockholm, Sweden.

Feldman, R., & Dagan, I. (1995). Knowledge discovery in textual databases (KDT). In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pp. 112–117 Montreal, Canada.

Feldman, R., Fresko, M., Hirsh, H., Aumann, Y., Liphstat, O., Schler, Y., & Rajman, M. (1998). Knowledge management: A text mining approach. In Reimer, U. (Ed.), *Proceedings of Second International Conference on Practical Aspects of Knowledge Management (PAKM-98)*, pp. 9.1–9.10 Basel, Switzerland.

Feldman, R., & Hirsh, H. (1996). Mining associations in text in the presence of background knowledge. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pp. 343–346 Portland, OR.

Fellbaum, C. D. (1998). *WordNet: An Electronic Lexical Database.* MIT Press, Cambridge, MA.

Freitag, D. (1998a). Information extraction from HTML: Application of a general learning approach. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 517–523 Madison, WI. AAAI Press / The MIT Press.

Freitag, D. (1998b). Multi-strategy learning for information extraction. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-98)*, pp. 161–169 Madison, WI.

Freitag, D., & Kushmerick, N. (2000). Boosted wrapper induction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pp. 577–583 Austin, TX. AAAI Press / The MIT Press.

Ghani, R., & Fano, A. E. (2002). Using text mining to infer semantic attirbutes for retail data mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM-2002)*, pp. 195–202 Maebash City, Japan. IEEE Computer Society.

Ghani, R., Jones, R., Mladenić, D., Nigam, K., & Slattery, S. (2000). Data mining on symbolic knowledge extracted from the Web. In Mladenić, D. (Ed.), *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining*, pp. 29–36 Boston, MA.

Grobelnik, M. (Ed.). (2001). *Proceedings of IEEE International Conference on Data Mining (ICDM-2001) Workshop on Text Mining (TextDM'2001)*, San Jose, CA.

Grobelnik, M. (Ed.)., text mining (2003). *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence(IJCAI-2003) Workshop on Text Mining and Link Analysis (TextLink-2003)*, Acapulco, Mexico.

Gusfield, D. (1997). *Algorithms on Strings, Trees and Sequences.* Cambridge University Press, New York.

Haase, K. B. (2000). Interlingual brico. *IBM Systems Journal*, *39*(3 / 4), 589–596.

Hahn, U., Romacker, M., & Schulz, S. (2002). Creating knowledge repositories from biomedical reports: The MEDSYNDIKATE text mining system. In *Proceedings of the 7th Pacific Symposium on Biocomputing*, pp. 338–349 Kauai, HI.

Hammond, K., Burke, R., Martin, C., & Lytinen, S. (1995). FAQ-Finder: A case-based approach to knowledge navigation. In *Working Notes of AAAI Spring Symposium: Information Gathering from Heterogeneous, Distributed Environments*, pp. 69–73 Stanford University. AAAI Press.

Han, J., & Kamber, M. (2000). *Data Mining: Concepts and Techniques.* Morgan Kaufmann, San Francisco.

Harabagiu, S. M., Bunescu, R. C., & Maiorano, S. J. (2001). Text and knowledge mining for coreference resolution. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-2001)*, pp. 55–62 Pittsburgh, PA.

Hearst, M. A. (1999). Untangling text data mining. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL-99)*, pp. 3–10 College Park, MD.

Hearst, M. A. (2003). What is text mining?. http://www.sims.berkeley.edu/~hearst/text-mining.html.

Henze, N., & Nejdl, W. (2002). Knowledge modeling for open adaptive hypermedia. In De Bra, P., Brusilovsky, P., & Conejo, R. (Eds.), *Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, pp. 174–183 Malaga, Spain. Springer Verlag. LEcture Notes in Computer Science Vol. 2347.

Hernández, M. A., & Stolfo, S. J. (1995). The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD-95)*, pp. 127–138 San Jose, CA.

Hotho, A., Staab, S., & Stumme, G. (2003). Ontologies improve text document clustering. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM-03)*, pp. 541–544 Melbourne, FL. IEEE Computer Society. Short Paper.

Kelin, D., Smarr, J., Nguyen, H., & Manning, C. D. (2003). Named entity recognition with character-level models. In Daelemans, W., & Osborne, M. (Eds.), *Proceedings of Conference on Computational Natural Language Learning (CoNLL-2003)*, pp. 180–183 Edmonton, Canada. ACM.

Kohonen, T. (1997). *Self-Organizing Maps* (2nd edition). Springer-Verlag, Berlin, Germany.

Kushmerick, N. (Ed.). (2001). *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001) Workshop on Adaptive Text Extraction and Mining*, Seattle, WA. AAAI Press.

Lamirel, J.-C., & Toussaint, Y. (2000). Combining symbolic and numeric techniques for DL contents classification and analysis. In *Proceedings of the First DELOS Network of Excellence Workshop on Information Seeking, Searching and Quering in Digital Libraries* Zurich, Switzerland.

Lavrac, N., & Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood.

Lenat, D. B. (1997). Cyc: A large-scale investment in knowledge infrastructure. *Communications of the Association for Computing Machinery*, *38*(11), 32–38.

Lent, B., Agrawal, R., & Srikant, R. (1997). Discovering trends in text databases. In Heckerman, D., Mannila, D. P. H., & Uthrysamy, R. (Eds.), *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pp. 227–230 Newport Beach, CA.

Leroy, G., Chen, H., & Martinez, J. D. (2003). A shallow parser based on closed-class words to capture relations in biomedical text. *Journal of Biomedical Informatics (JBI)*, *36*, 145–158.

Levenshtein, V. I. (1966). Binary codes capable of correcting insertions and reversals. *Soviet Physics Doklady*, *10*(8), 707–710.

Lin, D., & Pantel, P. (2001). Discovery of inference rules for question answering. *Natural Language Engineering*, *7*(4), 343–360.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. In Piatetsky-Shapiro, G., Agrawal, R., & Stolorz, P. E. (Eds.), *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pp. 80–86 New York, NY. AAAI Press.

Loh, S., Wives, L. K., & de Oliveira, J. P. M. (2000). Concept-based knowledge discovery in texts extracted from the Web. *SIGKDD Explorations*, *2*(1), 29–39.

Lucarella, D. (1988). A document retrieval system based on nearest neighbour searching. *Journal of Information Science*, *14*, 25–33.

Mannila, H., Toivonen, H., & Verkamo, A. I. (1997). Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, *1*(3), 259–289.

Manning, C. D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

McCallum, A., & Jensen, D. (2003). A note on the unification of information extraction and data mining using conditional-probability, relational models. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data* Acapulco, Mexico.

McCallum, A., & Nigam, K. (1998). A comparison of event models for naive Bayes text classification. In *Papers from the AAAI-98 Workshop on Text Categorization*, pp. 41–48 Madison, WI.

McCallum, A., Nigam, K., Rennie, J., & Seymore, K. (2000a). Automating the construction of internet portals with machine learning. *Information Retrieval*, *3*(2), 127–163.

McCallum, A., Nigam, K., & Ungar, L. (2000b). Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000)*, pp. 169–178 Boston, MA.

McCray, A. T., & Aronson, A. R. (2002). Automated and semi-automated indexing.. http://ii.nlm.nih.gov/resources/MTI_091102.pdf.

Mitchell, T. (1997). *Machine Learning.* McGraw-Hill, New York, NY.

Mladenić, D. (Ed.). (2000). *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining*, Boston, MA.

Monge, A. E., & Elkan, C. (1996). The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pp. 267–270 Portland, OR.

Monge, A. E., & Elkan, C. P. (1997). An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD 1997 Workshop on Research Issues on Data Mining and Knowledge Discovery*, pp. 23–29 Tuscon, AZ.

Mooney, R. J., & Roy, L. (2000). Content-based book recommending using learning for text categorization. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pp. 195–204 San Antonio, TX.

Muresan, S., & Klavans, J. L. (2002). A method for automatically building and evaluating dictionary resources. In *Proceedings of the 3rd International Conference on Language Resources & Evaluation (LREC-2002)* Las Palmas, Spain.

Muslea, I. (1999). Extraction patterns for information extraction tasks: A survey. In Califf, M. E. (Ed.), *Papers from the Sixteenth National Con-*

ference on Artificial Intelligence (AAAI-99) Workshop on Machine Learning for Information Extraction Orlando, FL. AAAI Press.

Muslea, I. (Ed.). (2004). *Papers from the AAAI-2004 Workshop on Adaptive Text Extraction and Mining (ATEM-2004) Workshop*, San Jose, CA. AAAI Press.

Muslea, I., Minton, S., & Knoblock, C. A. (1999). A hierarchical approach to wrapper induction. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pp. 190–197 Seatttle, WA. ACM.

Nahm, U. Y., Bilenko, M., & Mooney, R. J. (2002). Two approaches to handling noisy variation in text mining. In *Papers from the Nineteenth International Conference on Machine Learning (ICML-2002) Workshop on Text Learning*, pp. 18–27 Sydney, Australia.

Nahm, U. Y., & Mooney, R. J. (2000). Using information extraction to aid the discovery of prediction rules from texts. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining*, pp. 51–58 Boston, MA.

Nahm, U. Y., & Mooney, R. J. (2001). Mining soft-matching rules from textual data. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pp. 979–984 Seattle, WA.

Navarro, G., & Baeza-Yates, R. (1998). A practical q-gram index for text retrieval allowing errors. *CLEI Electronic Journal*, *1*(2).

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, *48*, 443–453.

Pereira, F. C. N., Tishby, N., & Lee, L. (1993). Distributional clustering of English words. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL-93)*, pp. 183–190 Columbus, Ohio.

Peterson, J. L. (1976). Computation sequence sets. *Journal of Computer and Systems Sciences*, *19*(1), 1–24.

Pierre, J. M. (2002). Mining knowledge from text collections using automatically generated metadata. In Karagiannis, D., & Reimer, U. (Eds.), *Proceedings of the Fourth International Conference on Practical Aspects of Knowledge Management (PAKM-2002)*, pp. 537–548 Vienna, Austria. Springer. Lecture Notes in Computer Sicnece Vol. 2569.

Pinto, D., McCallum, A., Wei, X., & Croft, W. B. (2003). Table extraction using conditional random fields. In *Proceedings of 26th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 235–242 Toronto, Canada. ACM.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo,CA.

Quinlan, J. R., & Cameron-Jones, R. M. (1993). FOIL: A midterm report. In *Proceedings of the European Conference on Machine Learning*, pp. 3–20 Vienna.

Riloff, E. (1993). Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 811–816 Washington, D.C.

Riloff, E. (1996). Using learned extraction patterns for text classification. In Wermter, S., Riloff, E., & Scheler, G. (Eds.), *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, pp. 275–289. Springer, Berlin.

Robertson, J., Wong, W. Y., Chung, C., & Kim, D. K. (1998). Automatic speech recognition for generalised time based media retrieval and indexing. In *Proceedings of the 6th ACM International Conference on Multimedia*, pp. 241–246 Bristol, England. ACM.

Salton, G. (1989). *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*. Addison-Wesley.

Sankoff, D., & Kruskal, J. B. (Eds.). (1983). *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison*. Addison-Wesley.

Savasere, A., Omiecinski, E., & Navathe, S. B. (1998). Mining for strong negative associations in a large database of customer transactions. In *Proceedings of the 14th International Conference on Data Engineering (ICDE-98)*, pp. 494–502 Orlando, FL. IEEE Computer Society.

Scheffer, T., & Leser, U. (Eds.). (2003). *Proceedings of the ECML/PKDD 2003 Workshop on Data Mining and Text Mining for Bioinformatics*, Dubrovnik, Croatia.

Schulz, K. U., & Mihov, S. (2002). Fast string correction with Levenshtein automata. *International Journal of Document Aanlysis and Recognition*, *5*(1), 67–85.

Schwartz, A. S., & Hearst, M. A. (2003). A simple algorithm for identifying abbreviation definitions in biomedical text. In *Proceedings of the 8th Pacific Symposium on Biocomputing*, pp. 451–462 Lihue, HI.

Scott, S., & Matwin, S. (1998). Text classification using WordNet hypernyms. In Harabagiu, S. (Ed.), *Proceedings of the COLING/ACL Workshop on Usage of WordNet in Natural Language Processing Systems*, pp. 38–44 Montreal, Quebec, Canada. ACL / Morgan Kaufmann Publishers.

Soderland, S. (1999). Learning information extraction rules for semi-structured and free text. *Machine Learning*, *34*, 233–272.

Soderland, S., Etzioni, O., Shaked, T., & Weld, D. S. (2004). The use of Web-based statistics to validate information extraction.. Papers from the AAAI-2004 Workshop on Adaptive Text Extraction and Mining (ATEM-2004) Workshop, San Jose, CA.

Srihari, R., & Li, W. (1999). Information extraction supported question answering. In *Proceedings of the Eighth Text REtrieval Conference(TREC-8)* Gaithersburg, MD. NIST Special Publication.

Srikant, R., & Agrawal, R. (1995). Mining generalized association rules. In *Proceedings of the 21st International Conference on Very Large Databases (VLDB-95)*, pp. 407–419 Zurich, Switzerland.

Srikant, R., & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In Apers, P. M. G., Bouzeghoub, M., & Gardarin, G. (Eds.), *Proceedings of the Third International Conference on Extending Database Technology (EDBT-96)*, pp. 3–17 Avignon, France. ACM. Lecture Notes in Computer Science Vol. 1057.

Stevenson, M., & Ciravegna, F. (2003). Information extraction as a Semantic Web technology: Requirements and promises. In Ciravegna, F., & Kushmerick, N. (Eds.), *Papers from the ECML/PKDD-2003*

Workshop on Adaptive Text Extraction and Mining Cavtat-Dubrovnik, Croatia.

Sullivan, D. (2000). The need for text mining in business intelligence.. DM Review.

Thompson, C. A., Smarr, J., Nguyen, H., & Manning, C. D. (2003). Finding educational resources on the Web: Eexploiting automatic extraction of metadata. In Ciravegna, F., & Kushmerick, N. (Eds.), *Papers from the 14th European Conference on Machine Learning(ECML-2003) and the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases(PKDD-2003) Workshop on Adaptive Text Extraction and Mining* Cavtat-Dubrovnik, Croatia.

Turtle, H. R., & Flood, J. (1995). Query evaluation: Strategies and optimizations. *Information Processing and Management*, *31*(6), 831–850.

Uchida, H., Zhu, M., & Della, S. (2000). *UNL: A Gift for a Millennium.* The United Nations University, Tokyo, Japan.

University of Southern California, I. S. I. (1998). RISE: A repository of online information sources used in information extraction tasks. http://www.isi.edu/info-agents/RISE/index.html.

White, M., Cardie, C., Han, C.-H., Kim, N., Lavoie, B., Palmer, M., Rambow, O., & Yoon, J. (2000). Towards translingual information access using portable information extraction. In *Proceedings of the Applied Natural Language Processing and the North American Chapter of the Association for Computational Linguistics (ANLP/NAACL-2000) Workshop on Embedded MT Systems*, pp. 31–37 Seattle, WA.

Winkler, W. E. (1999). The state of record linkage and current research problems. Tech. rep., Statistical Research Division, U.S. Census Bureau, Washington, DC.

Witten, I. H., & Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, San Francisco.

Wu, X., Zhang, C., & Zhang, S. (2002). Mining both positive and negative association rules.. pp. 658–665 Sydney, Australia. Morgan Kaufmann.

y Gómez, M. M., Gelbukh, A. F., & López-López, A. (2001). Text mining as a social thermometer. In Feldman, R. (Ed.), *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99) Workshop on Text Mining: Foundations, Techniques and Applications*, pp. 103–107 Stockholm, Sweden.

Yang, Y. (1999). An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval*, *1*(1/2), 67–88.

Yang, Y., Carbonell, J. G., Brown, R., Pierce, T., Archibald, B., & Liu, X. (1999). Learning approaches for detecting and tracking news events. *IEEE Intelligent Systems*, *14*(4), 32–43.

Yin, X., & Han, J. (2003). CPAR: Classification based on predictive association rules. In Barbará, D., & Kamath, C. (Eds.), *Proceedings of Third SIAM International Conference on Data Mining(SDM-2003)* San Francisco, CA. SIAM.

Zaki, M. J. (2002). Efficiently mining frequent trees in a forest. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, pp. 71–80 Edmonton, Alberta, Canada. ACM.

Zelikovits, S., & Hirsh, H. (2002). Integrating background knowledge into nearest-neighbor text classification. In *Proceedings of the 6th European Conference on Case-Based Reasoning (ECCBR-2002)*, pp. 1–5 Aberdeen, Scotland, U.K. Springer Verlag. Lecture Notes in Computer Science Vol. 2416.

Zhang, K., & Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, *18*(6), 1245–1262.