Annotating Web Tables Using Surface Text Patterns

by

Andong Wang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

While the World Wide Web has always been treated as an immense source of data, most information it provides is usually deemed unstructured and sometimes ambiguous, which in turn makes it unreliable. But the web also contains a relatively large number of structured data in the form of tables, which are constructed elaborately by human. Unfortunately, each relational table on the Web carries its own "schema". The semantics of the columns and the relationships between the columns are often ill-defined; this makes any machine interpretation of the schema difficult and even sometimes impossible.

We study the problem of annotating Web tables where given a table and a set of relevant documents, each describing or mentioning the element(s) of a row, the goal is to find surface text patterns that best describe the contexts for each column or combinations of the columns. The problem is challenging because of the number of potential patterns, the amount of noise in texts and the numerous ways rows can be mentioned. We develop a 2-stage framework where candidate patterns are generated based on sliding windows over texts in the first stage, and in the second stage, patterns are generalized and the redundant patterns are removed. Experiments are conducted to evaluate the quality of the annotations in comparison to human annotations.

Acknowledgements

First, I would like to thank my supervisor Dr. Davood Rafiei for the continuous support and patience with me. With his expertise, he gave me a lot of valuable advice.

Also, I would like to thank my family for their love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Web may be viewed as a huge corpus of unstructured documents, but a large volume of well-structured information is contained in these documents as well, and this provides an abundant source of what we refer to as relational data. This relational data is usually presented in the form of tables, each consisting of data values in the form of a two dimensional grid. An example is Table 1.1 with three rows and three columns, listing the birth year and the death year. The Web offers a corpus of over 100 million such tables on a wide range of topics [4]. Each Web table can be viewed as a single relational

Mozart	1756	1791
Einstein	1879	1955
Alan Turing	1912	1954

Table 1.1: Example Table

table providing factual and relational information. Some of these tables are elaborately designed and populated by knowledgeable people, describing useful facts and relationships.

The wide-spread use of tables for presenting data, in general, may be traced back to their high information density. Because of the semantic relations implied in a table layout and structure, only few descriptive words are needed for human to interpret semantics of the rows and the columns, and no additional information but relevant data entries (or entities) may be listed.

However, such structural information is not accessible to the machines. It lacks the metadata that is traditionally used in the interpretation of structured data. In Table 1.1, it may be obvious to human what the table records because of the listed values and the formatting. But, without help of any annotation or understanding of the table structure (metadata), deciphering this information can be challenging. This is the problem we study in this thesis, i.e. understanding table semantics through annotations.

A guiding principle in our work is that all rows in a table are expected to describe the same relationships between the columns. For the same reason, each annotation describes a property or a relationship that is expected to hold for all rows of the table. We do not require a header row or any type information that may describe the content beneath it, and this makes the problem more general.

There are many different ways of expressing an annotation [1, 8, 14]. Predefined labels or classes from an ontology can be selected to indicate the entity types or the relationships between the types. On the other hand, meaningful text patterns can represent semantic relations as well. For example, the relation between the first and the second columns in Table 1.1 can either be indicated by predefined type label such as "*BirthYear*" or represented by a set of patterns such as "*FLD1 was born in FLD2*", "*FLD1's birth year is FLD2*" where *FLD1* and *FLD2* respectively denote the first and the second columns of the table[1]. There are some shortcomings for the predefined label approach which will be discussed in Chapter 2.

In this thesis, we use text patterns to annotate web tables and to describe the semantics of the columns. Text patterns can provide useful information that can assist other processing techniques [9, 2]. Given relevant documents of a web table, where terms and entities of the table are discussed, we identify informative contexts surrounding the data entities in these documents. Some of the problems to be addressed here are (1) identifying the patterns, (2) ensuring their qualities and (3) selecting only a few patterns that best describe the relationships. The evaluation in our work is done by human annotators.

Textual contexts (or patterns) identified for a table can be used as meta-

[1]The content of a cell can be a sequence of words. If the sequence contains multiple words, it can be broke down into parts. More details on this can be found in Chapter 4.

data, allowing better use of tables in other applications. For example, this metadata can benefit information retrieval tasks over a corpus of tables. In particular, the content of a table can be indexed according to its semantic context, as described by the annotations. Search engines may use this information, for example, in detecting entities that share similar contexts or belong to the same class. Even for a general purpose search, the metadata can help better integrate tables into the search results. In the example of Table 1.1, knowing the relation between the first column and the second column would be useful when user searches "*Mozart birth year*". The keyword "*birth*" and "*year*" are expected to be mentioned frequently in the patterns linking the first and the second columns, and "*Mozart*" is an entity in the first column. In that case, the table may be regarded as closely related and returned as a result. Close to 30 million results containing discovered tables are clicked on Google within one day [4], which indicates that tables are a useful source of information to search engine users. Metadata can also assist in additional processing on table data. Two tables may share not many common data entries but express the same relationships between their columns. Such cases may be used, for example, in entity linking and entity resolution.

A challenging part of our approach, as the example shows, is that a given semantic meaning may be expressed using multiple text patterns, and a small change in an unimportant part of a pattern can result in another pattern. Two patterns can have common substrings or be substrings of a larger pattern and those patterns may or may not describe the same relationships. These factors contribute to redundancy in text patterns.

1.1 Thesis Statement

Our thesis statement is that high-quality textual contexts can be used to express the semantic relations between or within the columns of a table, and that text patterns are viable annotations for relational tables on the Web.

1.2 Thesis Contribution

The contributions of this thesis are as follows:

- Based on the observation that text patterns can represent a semantic meaning, we propose an unsupervised framework which annotates a web table with text patterns.
- We develop processing steps to enhance the quality of the text patterns, based on generalization and filtering.
- We study the problem of ranking the patterns, with different parameter settings evaluated and their performance compared.
- We evaluate our work on some real-world datasets, crawled from the Web, and against the human annotators.

1.3 Thesis Organization

The remainder of this thesis is organized as follows. We review related work in Chapter 2 and discuss their relationships to ours. Chapter 3 defines the problem and gives an overview of our framework. Chapter 4 introduces the first step, extracting text patterns from input documents, and some basic processing on the documents and extracted patterns. Chapter 5 presents methods for generalizing text patterns; also some notion of pattern quality is introduced and evaluated. In Chapter 6, we study the problem of eliminating patterns that are considered redundant. Chapter 7 studies the problem of pattern ranking. In Chapter 8 we summarize our thesis and discuss the future work.

Chapter 2

Literature Review

In this chapter, we review the literature closely related to ours. Our review includes the approaches on annotating tables with predefined labels, where the labels can be either from a handcrafted ontology or extracted from a corpus including the Web. The literature on sequential pattern mining is related in that it deals with the problem of extracting frequent sequences, similar to ours. We also review some utilization of text patterns on Question-Answering and Semantic Relation Extraction to express relationships.

2.1 Annotating Web Tables

Annotating tables can be viewed as a classification task where the goal is to classify the rows and columns into different categories. Adelfio and Samet [1] utilize similarities and differences between nearby rows to extract table schema. In their work, rows of a table may not describe the same relationships, which is fundamentally different from ours. A broad set of row classes is defined to label each row with a row functions (e.g data or header), and row features are manually designed to preserve the formatting and structure information (e.g. capital or not, numeric or not). As a final step, a Conditional Random Field (CRF) based classifier is trained with human labeled data and is used to classify each row. CRF is designed to maximize the probability of a row label sequence Y given row sequence X, i.e. $P(Y|X)$, where each row in X is represented by a feature vector. Compared to our work, this work focuses on the functions of a row, and the breadth of the designed row classes can affect the accuracy of the method. Also, it is not clear how useful the row labels are, and they may be better utilized in combination with other annotations.

Limaye et al. [8] leverage an existing type hierarchy with binary relations and entities that are instances of types. The authors annotate a table by associating cells with entities, columns with types and pairs of columns with binary relations between types. Different sets of features and similarity functions are utilized to compute scores for all the assignments over the table. Then, the authors use the joint inference over these scores to compute the probabilities for all assignment combinations, boosting the quality of assigned labels. The assignment combination with the highest probability is returned as the annotation for the table.

The same kind of annotation is done by Mulwad et al. [10]. In their approach, to predict a semantic class that characterizes a column, a ranked list of classes is first retrieved by submitting complex queries about the cells to a knowledge base. Thus a score can be computed for each class and cell string pair, (c_i, s_j), based on the rank of class c_i for cell string s_j and its predicted-Page Rank [13]. Then the class label that maximizes the score over the entire column is chosen as the column label.

Similarly, Venetis et al. [14] leveraged two databases for their labeling, and label the columns and the relations in a table. In this work, the two databases are extracted from the Web, instead of from an ontology, and the annotations are used to facilitate table search in their follow-up experiments. To label a column or a relation, they use a maximum likelihood method, which computes the probability of values $\{v_1, \ldots, v_n\}$ in a column having label l_i as follows

$$Pr\left[v_1, \ldots, v_n | l_i\right] = \prod_j \frac{Pr\left[l_i | v_j\right] \times Pr\left[v_j\right]}{Pr\left[l_i\right]} \propto \prod_j \frac{Pr\left[l_i | v_j\right]}{Pr\left[l_i\right]}$$

where $Pr\left[l_i\right]$ and $Pr\left[l_i | v_j\right]$ are computed based on the scores associated with the value and label pair (v_j, l_i). The labeling method attaches a class label to a column if sufficient number of values in the column are identified with that label in the database of class labels and analogously for binary relationships.

Unlike ours, these approaches do not handle nor produce surface text patterns to annotate a table. In addition, these approaches focus on binary relations whereas in our framework relations can have more than two columns.

2.2 Sequential Pattern Mining

Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, sequential pattern mining aims at finding frequent subsequences, i.e. subsequences whose frequency is no less than a minimum support threshold specified by user. If we treat a document as a sequence of words, then the existing algorithms for sequential pattern mining, such as PrefixSpan, are applicable to our problem.

Sequential pattern mining algorithms at first finds the set of frequent patterns of length one, $P = \{a, b, c, \ldots, n\}$. In our case, those will be patterns that have a single word. Next, the sequence database is shrank based on the frequent patterns and a new frequent item set $I = \{i_1, i_2, i_3, \ldots, i_n\}$ in the shrunk database is obtained. The next step is to check if the catenation of each pattern in P with each item in I is a valid pattern[1] and store the newly discovered length-two patterns in another set P'. Then the database is shrunk based on P' and the whole procedure is repeated to grow the pattern length by one. Sequential pattern mining algorithms shrink the size of the database in various ways and the choice of shrinking methods affects the performance the most [11].

However, there are major differences between surface text patterns and sequential patterns. First, sequential patterns can have arbitrary gaps between the terms whereas the size of the gap (if any) in a surface text pattern is more controlled, with sentences and phrases retaining their structure. For instance, the pattern "*is the of*" is constructed by frequent words and probably a sequential pattern with high frequency. However, there is no phrases or sentence structure in it and such a pattern is not valid in our case. Even if the gap constraints can be added to a sequential pattern mining algorithm to maintain the structure, those algorithms are not efficient when the minimum support is

[1]The result of such catenation is $(a, i_1), (i_1, a), \ldots, (b, i_1), (i_1, b), \ldots, (k, i_n), (i_n, k)$, where (a, i_1) means the pattern constructed by concatenating pattern a and item i_1

low, simply because not many patterns will be pruned. In particular, the minimum support for surface text patterns is usually low (e.g. in the range of 2), hence a large portion of the vocabulary would be considered frequent. Moreover, to get longer patterns in these algorithms, the whole database (whether projected or pruned) needs to be scanned to get the frequent items (in our case, words) to be concatenated to the current patterns. And the number of times the database is shrank depends on the desired pattern length. Thus, it is a stretch to use sequential pattern mining algorithms as they do not well fit the problem of finding surface text patterns.

2.3 Learning Surface Text Patterns

Surface text patterns are studied and applied in many areas of information retrieval, and there have been some studies on learning surface text patterns.

Ravichandran and Hovy [12] studied the problem in the context of a question answering system. The input to their algorithm is a question phrase with an associated answer phrase. A query is formed by the keywords from the question and the answer phrases and is submitted to a search engine; the returned documents are used to identify frequent text patterns that link question words to answer words. In these patterns, the literal mentions are replaced with question and answer tags. These patterns are used to locate answers for new questions. In a follow-up work, Hovy et al. [7] associate text patterns with question answer types and attempt to replace answer types with more specific type markers. Also, to address the problem of generalization, Greenwood and Gaizauskas [6] utilize external resources (i.e. a gazetteer and a named entity tagger) to tag dates and locations. Thus, date and location mentions are replaced with class tags.

The aforementioned methods extract text patterns to catch the semantics between objects the same as we do. However, these methods usually handle exactly one question term and one answer term. Another drawback is excessive redundancy and the lack of a generalization.

Chapter 3

Problem Statement and System Overview

In this chapter, we introduce the problem and discuss some qualitative features of an expected result.

3.1 Problem Definition

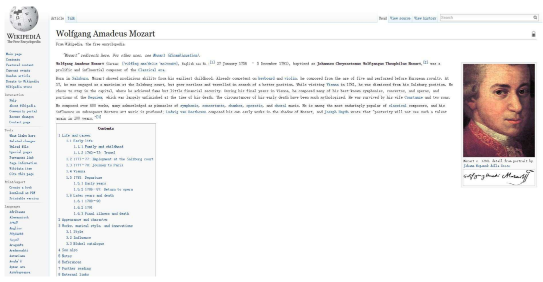
The problem can be formalized as follow:

> *Given a table T that consists of rows $\{r_1, r_2, \ldots, r_n\}$ and columns $\{c_1, c_2, \ldots, c_p\}$, for example, representing n data points and their feature values, and a set of documents or text segments $\{f_{11}, \ldots, f_{1m}, f_{21}, \ldots, f_{2m}, \ldots, f_{n1}, \ldots, f_{nm}\}$, where $f_{i1}, \ldots, f_{im}$ describe row r_i in the table, the goal is to extract surface text patterns that best describe the contexts in which a column is mentioned or which exhibit the inner relationships between multiple columns.*

An example of input data is Table 1.1, and a set of relevant documents describing the rows of the tables can be, for example, Wikipedia pages (as in Figure 3.1) or text fragments (as in Figure 3.2).

3.2 Dataset

Four datasets are used in our experimental evaluations. They are referred to as Homepage, Birth, Olympic and NBA, ordered based on the sizes of their documents from the smallest to the largest. The Homepage dataset consists



Wolfgang Amadeus Mozart

From Wikipedia, the free encyclopedia

"Mozart" redirects here. For other uses, see Mozart (disambiguation).

Wolfgang Amadeus Mozart (German: [ˈvɔlfɡaŋ amaˈdeːʊs ˈmoːtsaʁt], English see fn.;[1] 27 January 1756 – 5 December 1791), baptised as **Johannes Chrysostomus Wolfgangus Theophilus Mozart**,[2] was a prolific and influential composer of the Classical era.

Born in Salzburg, Mozart showed prodigious ability from his earliest childhood. Already competent on keyboard and violin, he composed from the age of five and performed before European royalty. At 17, he was engaged as a musician at the Salzburg court, but grew restless and travelled in search of a better position. While visiting Vienna in 1781, he was dismissed from his Salzburg position. He chose to stay in the capital, where he achieved fame but little financial security. During his final years in Vienna, he composed many of his best-known symphonies, concertos, and operas, and portions of the Requiem, which was largely unfinished at the time of his death. The circumstances of his early death have been much mythologized. He was survived by his wife Constanze and two sons.

He composed over 600 works, many acknowledged as pinnacles of symphonic, concertante, chamber, operatic, and choral music. He is among the most enduringly popular of classical composers, and his influence on subsequent Western art music is profound; Ludwig van Beethoven composed his own early works in the shadow of Mozart, and Joseph Haydn wrote that "posterity will not see such a talent again in 100 years."[3]

Contents

1 Life and career
  1.1 Early life
    1.1.1 Family and childhood
    1.1.2 1762–73: Travel
  1.2 1773–77: Employment at the Salzburg court
  1.3 1777–78: Journey to Paris
  1.4 Vienna
  1.5 1781: Departure
    1.5.1 Early years
    1.5.2 1786–87: Return to opera
  1.6 Later years and death
    1.6.1 1788–90
    1.6.2 1791
    1.6.3 Final illness and death
2 Appearance and character
3 Works, musical style, and innovations
  3.1 Style
  3.2 Influence
  3.3 Köchel catalogue
4 See also
5 Notes
6 References
7 Further reading
8 External links

Mozart c. 1780, detail from portrait by Johann Nepomuk della Croce

Figure 3.1: Wikipedia page for Wolfgang Amadeus Mozart

No other great composer kept so detailed a chronological list of their works as the catalogue compiled by Mozart during the last eight years of his life. It was begun in 1784 to bring order to his increasingly busy schedule of composing and performing. Mozarts meticulous notebook provides unique insight into the creation of some of historys most celebrated music.

Figure 3.2: Plain text describing Wolfgang Amadeus Mozart

of a table with 46 instance rows and 6 columns. Each row describes the contact information of a faculty member, e.g. office location, phone number and email address, in the department of Computing Science at the University of Alberta. Each faculty has a home page where the contact information is often mentioned; our dataset contains those pages as well. The Birth dataset describes the death year and the birth year relationships; the dataset consists of a table (similar to Table 1.1) with 17 rows and 3 columns. Each row is described by 3 documents. In Olympic dataset, the table has 36 rows and 4 columns, with each row introducing a host city for summer Olympic Games. NBA dataset consist of a table with 30 rows and 6 columns, listing the information related to NBA teams, e.g. their names and cities, etc. In the Olympic and the NBA datasets, each row of a table is described by one document and all the documents are from the same source, i.e. Wikipedia.

3.3 Desired Result

There are a few characteristics that can make a surface text pattern a "good" annotation.

First, the text patterns must be relevant; we expect the contexts surrounding the mentions of entries from the target table to be relevant. Accordingly, one may set a constraint that at least one entry from the target table must be mentioned in the resulting patterns. Second, the extracted text patterns must be representative. Uncommon patterns are less reliable to describe a column or a relationship. A pattern that is rarely seen or encountered is less likely to be a good representative, and because of its low frequency in documents, it is also less likely that the pattern can be utilized to locate similar entries. However, a pattern such as "*FLDi is*", where "*FLDi*" is a tag that can be replaced by a literal in column i, is expected to have a high frequency because of its short length and the commonness of the terms in spite of its low quality. Thus, the specificity of a pattern is also characteristic that should not be ignored. In this thesis, we study some methods for measuring the specificity and discuss some approaches to balance the generality and the specificity. Third, many patterns may not have enough support; however, these patterns can be small variations of each other. To collapse such patterns into more general patterns, we introduce the concept of gaps inside the textual contexts. In our case, gaps are wildcards that can represent one or more words. However, the introduction of wildcards requires setting constraints on the locations, the number of wildcards and the number of words a wildcard can replace. For example, a greedy replacement may not be a good idea since it can replace too many words, changing the meaning of the initial pattern.

3.4 System Overview

The input to our system is a table and a set of documents as described above, and the output is a set of patterns, ordered based on some notion of quality. We assume each document describes one and only one row in the target table. There can be multiple documents describing the same row but there cannot be one document associated with multiple rows. This is because if two rows are mentioned in the same document, patterns extracted from the document may describe relationships across rows; the space of such relationships is huge (quadratic on the size of the table) and annotating the relationships in this space is beyond the scope of this thesis.

Our system can be broken down to two stages.

- In the first stage, all candidate patterns are extracted and collected from the input set of documents, based on their co-occurrences with table entries.

- The quality of the patterns are enhanced in the second stage through generalization and filtering. Generalizing the patterns aims at increasing the coverage of patterns by collapsing similar patterns into a single one. Filtering aims at removing all redundant patterns.

We have built a system that does the aforementioned tasks, allowing different parameter settings by the user, and this system is used in our experiments. The parameters are introduced in Table A.1 of the Appendix A.

Chapter 4

Pattern Extraction And Processing

This chapter introduces the first processing stage, i.e. extracting the candidate patterns.

4.1 Pattern Extraction

Given a table and a set of documents that mention the table entries, each mention of a table entry gives rise to an annotation pattern. We want to collapse the patterns that match different table rows but otherwise the same. For this, we need some preprocessing and normalization on the documents as preparation work.

The first preparation is to replace all mentions of the table literals inside documents with tags that symbolize the columns and the part of the column they belongs to. It should be noted that the replacement is done word by word. For example, "*Turing*" is the second literal on the first column of Table 1.1, and inside all documents that describe the first row, the mentions of "*Turing*" are replaced by "*FLD1_2*". The first "1" in the tag indicates that the tag corresponds to column one. The second "2" means this tag represent the second part of the entry. There are some benefits in breaking cell strings into words. In particular, the relationship between different parts of a column can be caught and patterns mentioning only part of the cells will be extracted. Moreover, word-wise matching is easier for patterns to extract instances. A drawback is that the relationship between different columns as a whole is vague when only parts of the columns are displayed.

The second preparation involves HTML tags in Web pages. The choice of keeping or deleting HTML tags is subjective. Sometimes they are distractions in texts and may prevent us from finding "good" text patterns while in other times they can be part of a pattern. For example, HTML tags in "*FLD1_1 in ⟨a⟩ University of Alberta⟨/a⟩*" seem useful since they include an related entity. However, the tags in "*⟨font⟩FLD1_1 FLD1_2⟨/font⟩'s Home Page*" do not provide any useful information.

The third preparation focuses on text processing. All word are converted into low case. The text in the document is chunked into words or punctuation sequences.

After the preprocessing, the pattern extraction is done by placing a sliding window over text and extracting the candidate patterns (the steps are shown in Algorithm 1).

Algorithm 1 only extracts patterns containing the field tags to ensure that all candidate patterns are relevant. And a pattern's representativeness is measured in terms of the support the pattern has. For this, instead of using the number of mentions of a pattern in all documents, we use the number of distinct documents the pattern appears in. This will prevent taking a pattern because of its high frequency in a single document.

The lengths of the candidate patterns in the algorithm are controlled by the parameters **min** and **max**. In experiments, we normally set the minimum pattern length to 3, 4 or 5 and the maximum to 13, 14. **min** cannot be too small nor too large. A small **min** would lead to a large number of short patterns which may not be much descriptive, while a large **min** would make the algorithm miss some good short patterns. Similarly, a small value of **max** can result in a loss (of some useful patterns) and a large **max** can generate many patterns that are not expected to be frequent.

Regarding efficiency of Algorithm 1, there are two nested loops in Step 3-12, but this is not a performance bottleneck considering that **min** and **max** are small numbers (between 1 and 20). The document scanned only once in

Algorithm 1 Pattern Generation

Input:

F: a document after preprocessing

min,**max**: minimum and maximum pattern length

Output:

P: a set of patterns containing field tags with their occurrence

```
 1: Initialize P as empty set
 2: Set variable $last$ to 0                //$last$ records the index of latest field tag
 3: for $i = 0$ to $max$ do                 //Iterate over first $max$ words
 4:    for $j = min$ to $max$ do
 5:       if there exists field tag inside string $\mathbf{F}[i : i + j - 1]$ then
 6:          Add pattern $F[i : i + j - 1]$ to P
 7:          if the largest index of field tag $> last$ then
 8:             Set $last =$ the index of the field tag
                                            //Update $last$ to the latest field tag index
 9:          end if
10:       end if
11:    end for
12: end for
13: for $i = max + 1$ to length of F do     //Iterate over the rest of document
14:    if $\mathbf{F}[i]$ is a field tag then
15:       Set $last = i$                    //Update $last$ to the latest field tag index
16:    end if
17:    for $j = min$ to $max$ do            //Iterate for different pattern lengths
18:       if $i - j + 1 \leq last \leq i$ then //Check if $last$ is in the covered range
19:          Add pattern $F[i - j + 1 : i]$ to $P$
20:       end if
21:    end for
22: end for
```

Steps 13-22.

For long documents or a large number of documents to be processed, this algorithm can be naturally converted into a MapReduce [5] version since there is no global communication required during processing. Each mapper can scan its own chunk of documents (as in Algorithm 1) and generates key-value pairs, where a key is a pattern and value is the pattern frequency. This processing can miss boundary patterns that spread over two chunks. The ratio of missing patterns depends on the size of the chunks, and the bigger each chunk is, the smaller the ratio is. For Hadoop, with a data block size in the range of 64MB, loss ratio is negligible[1].

4.2 Basic Screening

Our pattern extraction, as discussed in the previous section, introduces some redundancy since whenever we identify a field tag, patterns of all possible lengths containing the tag are generated and this can produce some overlapping patterns. Here is an example. If the text "*he was born in FLD2_1*" exists in our pattern set, so is the text "*was born in FLD2_1*". We can use some simple heuristics to remove some of these patterns.

One heuristic is to remove patterns starting with or ending with common words. This is because there will be shorter variants of these patterns that exclude these redundant words. In our example, the pattern starting with "*he*" can be discarded. In general, we can probably exclude patterns starting with or ending with words belonging to connecting words or stop words, which refer to extremely common words of little significance on their own[2].

Similarly we remove patterns that start with or end with punctuations. We also require each pattern to contain at least one word or one informative punctuation besides the field tags, catching patterns like "*FLD1_1 (FLD2_1-*

[1]Assuming that a character takes one byte, a 64MB chunk can store 64×1024^2 characters. On average, each word contains 5, 6 letters and roughly $\left(64 \times 1024^2\right)/5 \approx 12 \times 1024^2$ words can be stored in a chunk. Patterns starting in the last *max* words will spread beyond the chunk boundary and cannot be detected. Thus the loss ratio is $max/\left(12 \times 1024^2\right)$.

[2]There is no single universal list of stop words used. The one we used is from `http://www.ranks.nl/stopwords`.

FLD3_1)", which can describe the birth-year and death-year relationship between the columns of Table 1.1.

Our last basic filtering is based on pattern support, or the number of documents that mention a pattern. We want each pattern to describe more than one row of the table; for the same reason, every pattern of frequency one can be dropped. Also if there are k documents that describe a single row, we can say that the frequency of a pattern must be large than k to describe a relationship in the table. That is, patterns of frequency k or less can be dropped.

Chapter 5

Generalizing Patterns

In this chapter we focus on the problem of pattern coverage and specificity.

5.1 Generating Generic Pattern

A typical approach to generalize a pattern is introducing wildcard symbols in the pattern text. But there are questions that need to be addressed, such as where to put a wildcards symbol, and how many wildcards should be introduced, etc.

The first question we want to tackle here is where these symbols should be. The most obvious option is putting no constraint on the location of a wildcard symbol. However, this would lead to the problem of over-generalization since this can collapse many unrelated patterns into one generic pattern. Considering that a wildcard symbol should not be too far from the field tags, a position constraint based on the distance to the field tags may be introduced. In our experiments, we assume the distance between wildcard symbols and field tags is at most one.

The next constraint is the number of words a wildcard symbol can represent or match. We make a conservative choice and in our experiments, we assume a wilcard can represent at most two words. The consideration is that we would rather not collapse less similar patterns than over-generalizing patterns, which would produce undescriptive generic patterns with a high frequency.

The last question is how many wildcards there should be in a pattern. Even though this is a parameter that can be set by user, we cannot think of many cases where more than 1-3 wild cards can be introduced without over-generalizing the patterns. Wildcard symbols aim at generalizing parts around field tags and extra wildcards can over-generalize the patterns.

For the generation of generic patterns, the whole pattern set is scanned and for each pattern, we replace parts satisfying the above conditions with wildcards to produce a generic pattern. For instance, when scanning "*FLDi was born in*", wildcard symbols replacing "*was*" and "*was born*" satisfy the above conditions. Thus, generic pattern "*FLDi * born in*" and "*FLDi ** in" are generated.

For each generic pattern that is introduced, the relationship between the generic pattern and the candidate patterns it covers or represents are kept. Treating the generic and the candidate patterns as two disjoint sets, the relationship between the two sets can be described using a bipartite graph with each edge showing if a candidate pattern is covered by a generic pattern. This is demonstrated in Figure 5.1 for two generic patterns and three candidate patterns they cover. Two generic patterns can cover the same candidate pattern(s), as shown in Figure 5.1.

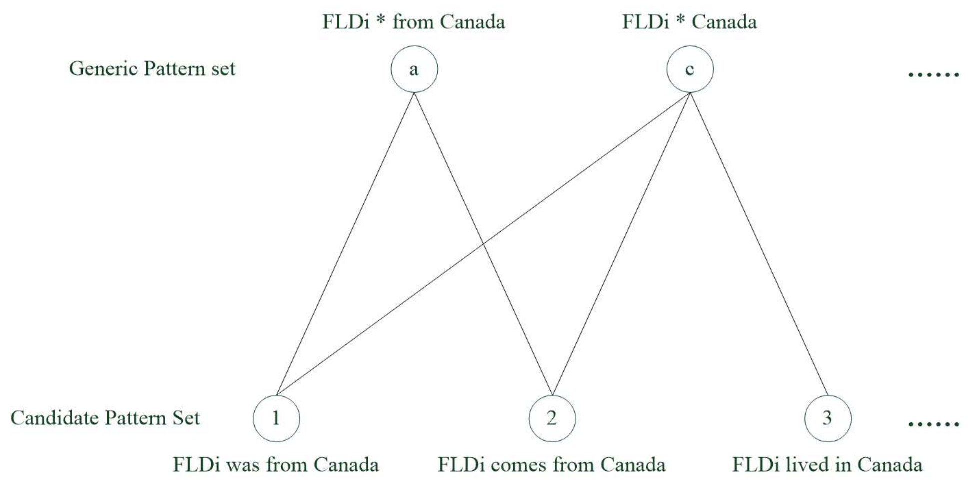


Figure 5.1: Demonstration of structure of generic and candidate pattern set and links between them

Since generic patterns are introduced in one scan of the candidate patterns and without a join between candidate patterns, a generic pattern may only cover one candidate pattern. Such patterns are redundant and provide no extra information; we want to filter out these patterns.

Next we discuss some qualitative measures to identify good generic patterns from the set.

5.2 Measuring Pattern Coverage and Specificity

Each candidate pattern can only cover itself. By introducing wildcard symbols (or gaps), which can match any words or continuous punctuation sequences, a pattern can cover more candidate patterns. For instance, there are patterns "*FLDi comes from FLDj*" and "*FLDi is from FLDj*". After introducing one wildcard symbol, we have "*FLDi * from FLDj*", which covers both candidate patterns. Through this generalization process, the coverage of a pattern set increases and its specificity declines. However, there should be a balance between the two. As we mentioned in Chapter 3, we can replace all words with wildcards except field tags. The replacement will maximize the coverage, but the resulting pattern may be of no use.

5.2.1 Specificity

We define the specificity of a pattern as the probability that a random text does not match the pattern, which equals to one minus the probability that it does match.

Based on the definition, we can measure the specificity of a pattern p, symbolized as $\mathit{specificity}(p)$. Let $Pr(p)$ denote the probability that a random text matches pattern p. Given that pattern p is a sequence of words, $\langle w_1, w_2, \ldots, w_n \rangle$, if we assume the matching of words are independent of each other, the probability of p matching a random text is the product of all probabilities of matching each compositional word. Thus, we have:

$$\mathit{specificity}(p) = 1 - Pr(p), \tag{5.1}$$

$$\text{where } Pr(p) = Pr(w_1) \times Pr(w_2) \times \cdots \times Pr(w_n). \tag{5.2}$$

Here $Pr(w)$ is the probability of a term w. Since wildcard symbols and field tags can match any text, their probability of a match is set to 1, e.g. $Pr\,(\textit{“*”}) = 1$ and $Pr\,(\textit{“FLDi”}) = 1$

Brin [3] uses another formula to measure the specificity, even though the underlying definition is similar to ours. The definition he gives is based on the log-likelihood that a uniformly distributed random variable matches a pattern. But, for quick computation, the formula for the specificity of pattern p is

$$\textit{specificity}\,(p) = |\text{p.prefix}||\text{p.middle}||\text{p.suffix}|$$

where |s| denotes the length of s. Since in his work, patterns allow only two entities (corresponding to our field tags), pattern p can be split into three parts, namely p.prefix, p.middle, p.suffix. This is a bit simplistic model that only cares about the number of characters in a pattern.

As for a set of patterns, a naive way to compute the specificity of the set is to use the mean or the median specificity. However, there is no good explanation for this. Instead, we generalize our definition of specificity to a set as follows. The specificity of a set of patterns is the probability that a random text does not match any pattern in the set.

If $S\colon\{\,p_1, p_2, \ldots, p_n\}$ denotes a set of patterns, the specificity of S can be expressed as

$$\begin{aligned} specificity(S) &= (1 - Pr\,(p_1)) \times (1 - Pr\,(p_2)) \times \cdots \times (1 - Pr\,(p_n)) \\ &= \textit{specificity}\,(p_1) \times \textit{specificity}\,(p_2) \times \cdots \times \textit{specificity}\,(p_n) \quad (5.3) \end{aligned}$$

where $Pr\,(p)$ can be computed as in Equation 5.2. The definition may seem problematic when comparing two pattern sets of different sizes. The set with more members may have a lower specificity even if the specificity of each of its members is higher than that of the other set, just because its size is larger. However, in our work, specificity is only compared between pattern sets derived from the the same set, and is used to demonstrate the difference between generalizing processes. Thus it is not a problem here.

5.2.2 Coverage

The coverage of a candidate pattern (without wildcards) is one since it can only cover itself, and the coverage of a generic pattern can be larger than one. However, instead of directly working with the number of covered patterns, we normalize it by dividing the number of patterns covered the set the patterns belong to. Thus the coverage for pattern p is

$$coverage\,(p) = \frac{number\ of\ patterns\ p\ covers}{number\ of\ patterns\ the\ set\ covers}. \tag{5.4}$$

The semantics expressed by all patterns in the set that p belongs to can be seen as a semantic space. Then the coverage of the pattern p can be alternatively interpreted as the portion of the semantic space that p covers.

The coverage of a set can be taken as the mean coverage of its members, and this gives a number between 0 and 1, which is comparable to the specificity.

5.3 Generalization Methods

Given a set of candidate patterns P_c and a set of generic patterns P_g, our goal is to select a set $P \subseteq (P_c \cup P_g)$ such that P covers the patterns in P_c and is more "concise". An algorithm to find P is to initially set $P = P_c$ and iteratively update it by replacing some of the patterns in P with more generic patterns in P_g. In the process, when adding a generic patterns, all covered candidate patterns are removed from P, and the selected generic pattern is also removed from P_g. The selection process aims at optimizing an objective function based on the coverage and the specificity (see Section 5.3.1 for details). Our next statement shows that the specificity either decreases or remains the same in each iteration.

Proposition 5.3.1. *Consider candidate patterns, $P_1, \ldots, P_n$ that are covered by a generic pattern P'. After replacing $P_1, \ldots, P_n$ with P', the specificity of the set cannot increase.*

Proof. Since generic pattern P' covers patterns $P_1, \ldots, P_n$, any random string that matches at least one of $P_1, \ldots, P_n$ must also match P'. Furthermore, any string that does not match any of $P_1, \ldots, P_n$ may still match P'. As a result, the probability that a random string does not match any of $P_1, \ldots, P_n$ is higher than or equal to the probability that it does not match P', i.e.

$$\begin{aligned} 1 - Pr\,(P') &\leq (1 - Pr\,(P_1)) \cdots (1 - Pr\,(P_n)) \\ \text{or} \quad \textit{specificity}\,(P') &\leq \textit{specificity}\,(P_1) \cdots \textit{specificity}\,(P_n) \end{aligned}$$

This completes the proof. □

It is noteworthy that the removal of covered candidate patterns in previous iterations can affect the coverage of other unselected generic patterns, because different generic patterns may cover the same candidate patterns. Take Figure 5.1 as an example. If we pick the generic pattern "*FLDi * Canada*" and add it to P, candidate patterns "*FLDi was from Canada*", "*FLDi comes from Canada*" and "*FLDi lived in Canada*" will be removed from P_c. Thus in the next iteration, the absolute coverage (not normalized) of pattern "*FLDi * from Canada*" will decrease by 2 since two of the candidate patterns it covers are already removed.

A question to be addressed now is how to pick the generic patterns. This is the problem we study next; we present some objective functions that optimize one or both of coverage and specificity.

5.3.1 Selecting Generic Patterns

We propose four methods based on specificity and coverage for selecting generic patterns:

Real Coverage Method This method focuses on the coverage. At each iteration, we pick the generic pattern currently covering the most candidate patterns. It is a greedy method and at each step we make a locally optimal choice. In our experiments, the process continues until in one iteration, the coverage for the selected pattern is one or all generic patterns are selected.

Potential Coverage Method This method is similar to the previous one with a difference that in this method, we compute the potential coverage in terms of the number of possible matches for a wildcard, instead of the real coverage. To compute the potential coverage, the coverage for each wildcard symbol is needed, which can be derived based on the number of possible text it can represent. In Figure 5.1, the wildcard in "*FLDi * Canada*" can replace a certain set of words, including "*was*" and "*comes*". As a result, the coverage for that wildcard in the particular generic pattern can be denoted as the size of the set of words it replaces. In this example, the coverage for the wildcard is 2. Now given a generic pattern p with wild card symbols $s_1, s_2, \ldots, s_n$, let $set(s)$ denotes the set of text that s can replace. The potential (possible) coverage for p can be expressed as

$$p\_coverage(p) = |set(s_1)| \times |set(s_2)| \times \ldots \times |set(s_n)| \tag{5.5}$$

Sometimes the potential coverage is the same as the real coverage. However, when the real coverage becomes large, most of the time they will differ. The selection process is the same as before, with the generic pattern that has the most potential coverage selected in each iteration. In the experiments, the selection process stop when the potential coverage for the selected pattern is equal to 1 or there are no more patterns to be selected.

Specificity Method This method focuses on the specificity, instead of coverage. At each iteration, we select the most specific pattern in the set of generic patterns. The selection process continues until the coverage for the selected pattern equals to 1 or the generic pattern set is empty.

Unlike the methods discussed earlier, there is an additional update after each iteration to the set of generic patterns P_g. After each iteration, the generic patterns that now cover one candidate patterns must be removed. Otherwise, the selected generic pattern may cover only one pattern and the process stops, while there may be another pattern less specific, covering more than one pattern. In that case, the process should continue.

With the updating action, one of the stopping conditions in our experiments (i.e the coverage for a selected pattern is equal to 1) can never be true for this method. The process only stops when the generic pattern set is empty.

Balanced Method Taking both coverage and specificity into consideration, this method aims to find a balance between the two. As a result, instead of simply maximizing one of them, the area below the coverage-specificity curve is maximized. Suppose after the i-1th selection process, the coverage and the specificity values are c_{i-1},s_{i-1}; this pair forms a data point on the coverage-specificity curve. In the next selection process, a generic pattern set containing n different patterns will provide n different coverage and specificity value pairs or points, $(c_{i1}, s_{i1}), \ldots, (c_{in}, s_{in})$. From these n points, we want to find the point, denoted as $(c_i,\ s_i)$, that maximizes the area of a trapezoid formed by data points (c_{i-1}, s_{i-1}), (c_i, s_i) and points on axes $(c_{i-1}, 0)$, $(c_i, 0)$. The area for the trapezoid can be computed as $(c_i - c_{i-1})(s_i + s_{i-1})/2$. The pair giving the maximum area is selected in each iteration.

Random Method This method can be used as a baseline. Each time we randomly choose a pattern from the generic pattern set until the set is empty. Since the selection is random, the problem with the stopping condition, as arises for the specificity method, also arises. There is also a need for filtering generic patterns that cover only one candidate pattern after each iteration. Consequently, the stopping condition is when the generic pattern set is empty.

5.3.2 Experiment

In our experimental evaluation, we wanted to see if there is a significant difference between the methods studied here and if one method finds a better balance between coverage and specificity

The performance of all five methods is tested on aforementioned four datasets (introduced in Section3.2) and are demonstrated in Figures 5.2 to 5.5. In the experiments, the maximum number of wildcard symbols per pattern is set to two. For Random method, each data point is a average of 5 runs.

Because of overlaps, some methods are not well-shown, but as the size of the datasets becomes larger and the number of patterns increase, the difference and trends become more obvious.

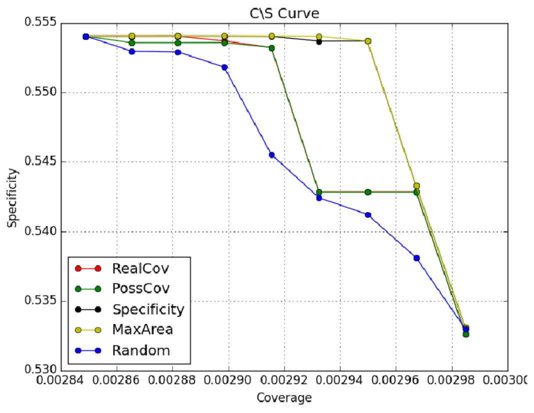


Figure 5.2: Specificity-Coverage curve for 5 methods on Homepage dataset

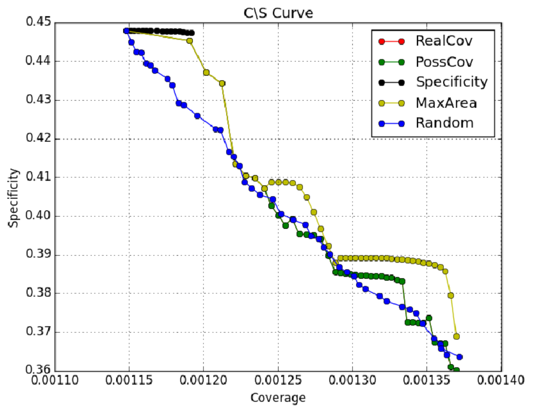


Figure 5.3: Specificity-Coverage curve for 5 methods on Birth dataset

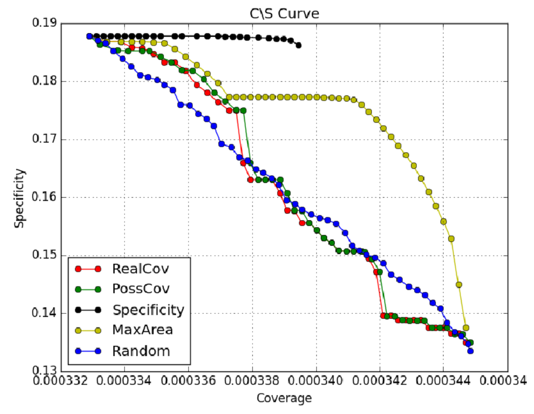


Figure 5.4: Specificity-Coverage curve of 5 methods on Olympic dataset

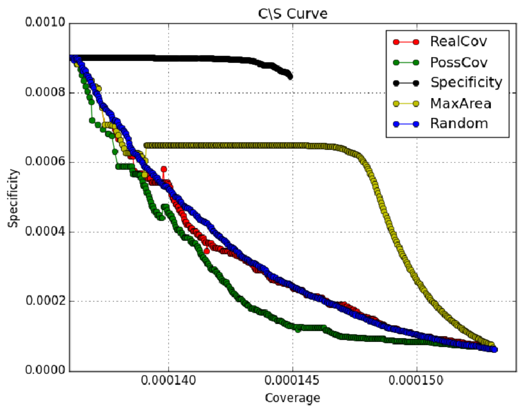


Figure 5.5: Specificity-Coverage curve for 5 methods on NBA dataset

5.3.3 Analysis and Comparison

Several observations can be made about the comparison. The first is about the nice performance of the balanced method. Even though it is a greedy method and makes locally optimal choices at each iteration, it shows a better performance on all four datasets, maintaining a high specificity while the coverage increases. The method does not seem to over-generalize or under-generalize candidate patterns, compared to the other methods. In our experiments, the coverage is pushed to its maximum to observe the performance of the methods. However, in practice, the generalizing process can stop earlier, for example, when the slope of the curve is getting big.

The next observation is the problem of over-generalization of the Possible Coverage method. This method is based on the number of combination that the wildcards can be assigned, instead of the real coverage. The method uses only partial coverage information and derives the rest of the information through an induction. For instance, consider candidate patterns "$A\ X\ B\ I\ C$", "$A\ Y\ B\ J\ C$", "$A\ Y\ B\ I\ C$" and generic pattern "$A\ *\ B\ *\ C$". Since the generic pattern can cover all three candidate pattern, its real coverage is 3. However, the first "$*$" in the generic pattern can be replaced by $\{X, Y\}$ and the second "$*$" can be replaced by $\{I, J\}$. Thus the possible coverage of the generic pattern is $2 \times 2 = 4$. We expect this to happen more often when the dataset size increases, leading to more candidate patterns. In Figures 5.2 and 5.3, the RealCov and PossCov almost perform the same. But in Figures 5.4 and 5.5, they totally differ. Particularly in Figure 5.5, the specificity drops dramatically when the coverage grows, which is an indication that PossCov favors coverage more than RealCov.

Another noticeable phenomenon is the strange performance of the Specificity method. Unlike other methods, the specificity maintains high specificity when the coverage grows. However, the coverage does not increase much compared to other methods. We will explain this in the context of the patterns shown in Figure 5.6.

There are two different ways of covering the nodes 1, 2 and 3 shown in the

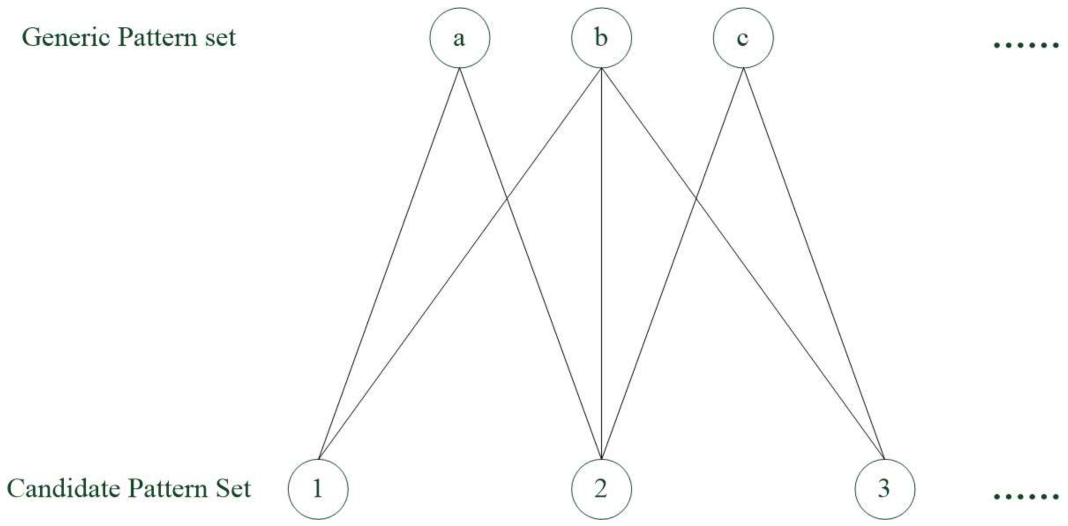


Figure 5.6: Example structure of candidate and generic pattern set

figure. One covering is {a,c} and another covering is {b}. Since node b covers more patterns than node a or c, node b's coverage is higher than the average coverage of nodes a and c, and its specificity must be less than or equal to the specificity of node a or c. As a result, for the pattern set, the covering $\{a, c\}$ can result in lower coverage and higher specificity than covering $\{b\}$. In our setting, the Specificity Method will select $\{a, c\}$, and this leads to a higher specificity and a lower coverage as, which is consistent with what is shown in Figures 5.2 to 5.5.

Chapter 6

Filtering Redundant Patterns

In the previous chapters, we have addressed the problem to generate and generalize the patterns. Our goal in this chapter is to eliminate patterns that provide no additional information. We did filter patterns of low frequency in Section 4.2, because uncommon patterns are not expected to carry much general semantic meaning. This type of filtering requires no comparison between the patterns and is merely based on the texts of the patterns and their frequencies. However, the filtering we enforce in this chapter requires comparisons between the patterns to identify the patterns that are redundant in the presence of other patterns.

6.1 Pattern Boundary Filtering

Our pattern generation can produce many overlapping patterns. In this processing step, the goal is to eliminate patterns that are unnecessarily long. However, a criterion to determine whether the length of a pattern is appropriate is required. From our observation, we propose a hypothesis and based on the hypothesis, a criterion is built.

6.1.1 Hypothesis

A pattern cannot be considered too long unless there are resembling patterns that are shorter. Thus, the criterion to remove such patterns must involve comparisons between the patterns.

Consider two patterns p_s and p_l such that p_s is a substring of p_l. When

p_s is frequent, there is a good chance that p_l is also frequent. We want to detect cases where p_l can be removed in favor of p_s. We observe that when a pattern spreads beyond its boundary, in most cases there is a sudden drop in its frequency. For instance, in our Homepage dataset, we observe that there are multiple versions of pattern "*phone*: (*FLD4_2*) *FLD4_3*−*FLD4_4*", which is at its ideal length, with a relatively high frequency. Here "*FLD4*" is the column for phone numbers and this pattern gives the format for phone numbers. However, there are other redundant patterns that are not at their ideal length, such as "*phone*: (*FLD4_2*) *FLD4_3*−*FLD4_4 fax*", which include unrelated text, or text that should belong to a different pattern. In this particular case, adding the term "*fax*" leads to a over 40% drop in the pattern frequency.

Based on this observation, we hypothesize that a sudden drop in frequency can be a sign of a meaningless extension. We have also considered the cases where unrelated text is at the front of a pattern, but in practice this seems to be a rare case. That is consistent with the fact that English is read left-to-right, and that the new content is always expected to be on the rightmost end of the patterns, which means one needs to check the new content and detect whether it is a continuation of the previous text. For the same reason, we only detect redundant text at the end of the patterns.

6.1.2 Boundary Detection using a Prefix Tree

Assuming that any redundant text is at the end of a pattern, for each redundant pattern, there must be a prefix pattern in the pattern set. A prefix tree can be constructed to efficiently detect the prefix relationships between patterns. The prefix tree also stores, for each node, the frequency of the pattern represented by the node. Figure 6.1 shows an example prefix tree.

Each edge in the tree is associated with a word or a sequence of punctuations, describing the text that must be seen to reach the next node, i.e. the children nodes. Each node stands for a particular pattern constructed by the text along the path to the node. Additionally, the frequencies of the patterns are stored in their corresponding nodes. If one pattern is not seen yet, its

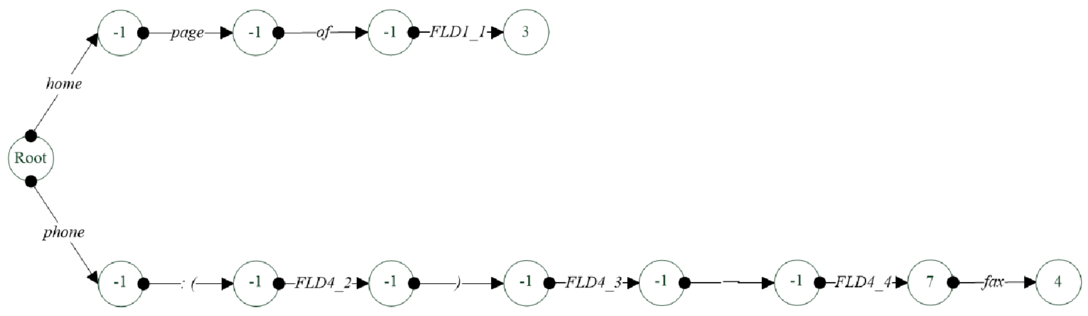


Figure 6.1: Example prefix tree

frequency is set to -1. In Figure 6.1, apart from two example patterns used in Section 6.1.1, there are only one more pattern, "*home page of FLD1*" with frequency 3.

Given a prefix tree of the patterns, we can traverse the tree and, for each node i, check the frequencies of all its children nodes $\{i_1, i_2, \ldots, i_n\}$ to detect if there is a rapid drop in frequency from node i to any of its children. If there is, the detected child node i_j can be considered redundant. However, the amount of the decline to call a pattern redundant needs to be determined experimentally[1]. Since the number of relevant documents can vary from one dataset to next, using an absolute value for the drop is not a reasonable option. Denote the frequency of node i with f_i and the frequency of a child node i_j with f_{i_j}. The pattern corresponding to node i_j can be called redundant if $f_{i_j}/f_i < t$ for some threshold t. In our next experiment, we seek some good values for the threshold t.

6.1.3 Experiments

In this section, we apply our boundary detection to the four datasets, Homepage, Birth, Olympic and NBA. Each pattern that is deemed redundant by our algorithm is examined by human annotators to detect if the pattern is truly redundant. The precision of a method is measured as the ratio of the number of true redundant patterns to the number of redundant patterns that

[1]Even though it can be treated as a parameter to be provided by users, we would like to set a reasonable value for it. Because our hypothesis should be general and the magnitude of a signal decline should be same across different types of patterns.

are detected.

To examine the detected patterns, a sample is drawn from the set of detected nodes and the patterns represented by the nodes in the sample are compared to the patterns represented by their parent nodes. Table 6.1 shows some examples of the detected patterns together with their parent patterns. The annotators were asked to focus on whether two patterns convey the same meaning, and mark patterns that they are certain to be redundant as a correct detection. In cases where annotators were not sure or the patterns being compared seemed ambiguous, the patterns were marked as an incorrect detection since we are not sure whether it should be kept. Example 5 in Table 6.1 is an example of such cases. The semantic meaning of two patterns in the Example 5 is not clear and it is marked as incorrect detection, because we are not sure if it is safe to discard the detected pattern.

Example 1	Detected pattern	*fax: (FLD4_2) FLD4_3-1071 email*
	Parent Pattern	*fax: (FLD4_2) FLD4_3-1071*
Example 2	Detected pattern	*in the olympic games - FLD4_1 FLD1_1 notes*
	Parent Pattern	*in the olympic games - FLD4_1 FLD1_1*
Example 3	Detected pattern	*FLD4_1 summer olympics in FLD1_1, FLD2_1 africa*
	Parent Pattern	*FLD4_1 summer olympics in FLD1_1, FLD2_1*
Example 4	Detected pattern	*basketball team based in FLD2_1 FLD1_2*
	Parent Pattern	*basketball team based in FLD2_1*
Example 5	Detected pattern	*[edit] olympics portal FLD4_1 summer*
	Parent Pattern	*[edit] olympics portal FLD4_1*

Table 6.1: Example detected patterns with their superstring pattern

Since the datasets Homepage and Birth were small, all detected redundant patterns were passed to the annotators. For the Olympic dataset, we took a 30% sample of the detected redundant patterns using reservoir sampling and for NBA, a 10% sample is used. The result is shown in Figure 6.2.

In our experiments, three different values for threshold t were tried: 60%, 50% and 40%, respectively corresponding to frequency drops of 40%, 50% and

Threshold	40%			50%			60%		
	True	Total	Precision	True	Total	Precision	True	Total	Precision
Homepage	15	17	88.23%	19	25	76%	19	33	57.58%
Birth	21	25	84%	26	33	78.79%	28	39	71.79%
Olympic	23	35	65.71%	36	61	59.02%	48	82	58.54%
NBA	38	61	62.3%	56	91	61.54%	62	101	61.39%

Table 6.2: Precision of detecting redundant patterns as the threshold on frequency drop varies

60%. For larger values of t, the constraint becomes more relaxed and more patterns are considered redundant. However, the precision decreases at the same time. Lower threshold values means stricter conditions and larger frequency criterion, which leads to fewer detected patterns and higher precision. In theory, this will also lead to a lower recall. But we do not have much knowledge of the total number of redundant patterns, and thus we are not able to measure the recall. Nevertheless, we prefer a high precision over a high recall. Because a high precision here means discarding a set of patterns most of which are truly redundant while leaving some redundant patterns undetected. A high recall means discarding a large set of patterns, some of which can be of good quality. In a filtering step, the retention of good patterns is more important than the removal of all unnecessary patterns.

From Table 6.2, when the size of a dataset becomes larger, the precision seems to drop overall, except for the largest dataset NBA. Its precision stabilizes around 62% and almost does not change for different thresholds.

Moreover, Table 6.2 confirms our hypothesis that for patterns in all four categories, a rapid drop in frequency in most cases signals an unrelated content at the end. The precision is higher than 50% in all experiments, which means the number of false positive is smaller than the number of correct predictions.

Another interesting observation in this experiment is that there were less than ten generic patterns reported to be redundant, out of all the patterns passed to the annotators, which means the generic patterns we generate in the last step contain less redundancy.

6.2 Substring Filtering

In Section 6.1, we studied the problem of redundancy for long patterns. In this step, on the other hand, we aim at removing short redundant patterns.

To demonstrate the concept, consider pattern "*phone*: (*FLD4_2*) *FLD4_3*–*FLD4_4*" and its substring "*phone*: (*FLD4_2*) *FLD4_3*–", which is one token shorter and provide no additional information.

In general, for any pattern p, its substring p' must also be in the pattern set as pattern p' as long as pattern p' is longer than the minimum length and not filtered in the previous steps. This is because if pattern p is frequent, its substrings must also be frequent. We need an approach to determine whether pattern p' provides any useful additional information.

6.2.1 Generality and the frequency of mentions

Since pattern p' is a substring of pattern p, any text that matches p must also match p'. As a result, the frequency of the mentions of p' must be higher than or equal to that of p. If a shorter pattern p' is truly more general than a longer pattern p, then the frequency of mentions of p' must be much larger than p. We can use this observation about the frequency of mentions to determine whether pattern p' is redundant in the presence of pattern p.

As mentioned, the frequencies of patterns p' and p need to be compared, and if the frequency of p' is close to that of p, then p' can be removed in favor of p. Otherwise, pattern $p\prime$ cannot be removed since is more general.

6.2.2 Substring Detection using a Suffix Tree

For an efficient implementation of our substring filtering, we need a data structure that can identify the substring relationships between the patterns. We adopt a suffix tree to detect such cases.

Unlike a prefix tree, a suffix tree needs to store each possible suffix for a pattern. For instance, for the pattern "*phone*: (*FLD4_2*) *FLD4_3*–*FLD4_4*", a prefix tree only needs to store the sequence of words. To insert the same pattern into a suffix tree, all its suffixes need to be generated and stored. The suffixes of this pattern include "*FLD4_3* − *FLD4_4*"[2], "*) FLD4_3* − *FLD4_4*", "*FLD4_2) FLD4_3* − *FLD4_4*", ": (*FLD4_2*) *FLD4_3* − *FLD4_4*" and itself. All these suffixes will be stored in the same way as prefix tree does, as shown in Figure 6.2. When one pattern is needed to be compared, we traverse the suffix tree to check if it is the prefix of one of the suffixes. Since in a suffix tree there is no information pointing out which pattern has the suffix we are dealing with, the frequency of each suffix needs be stored. However, compared to a prefix tree, there are two changes on the way the frequency data is stored. First, the frequency needs to be store in each node of the path leading to a suffix, rather than only in the node where the suffix is stored. Hence, whenever a substring relation is confirmed, the frequency of the substring pattern can be directly compared to that of the node where the traversal ends. Second, because two different patterns can share a suffix, a suffix can have multiple frequencies, each corresponding to a pattern. Thus, a set of frequencies needs to be stored in each node, instead of a single frequency.

In Figure 6.2, a suffix tree is shown, where all suffixes of patterns "*phone*: (*FLD4_2*) *FLD4_3*−*FLD4_4*" and "*phone*: (*FLD4_2*) *FLD4_3*−*FLD4_4 fax*" are in the tree. Since many suffixes share visiting paths, a number of nodes in the tree have two frequencies, $\{4, 7\}$. Furthermore, we do not want a pattern to match itself in the suffix tree when checking whether it is a substring. Thus, the node representing a pattern does not include the pattern's frequency. This is shown on the leftmost branches of the tree, where the second last node has frequency $\{4\}$ and the last node has the empty set. This modification avoids matching a pattern to itself.

Suppose we need to check whether pattern "*FLD4_2) FLD4_3*" is a substring of any pattern. We can use this pattern to traverse the tree. The end of the pattern is hit when it traverses along the path for "*FLD4_2) FLD4_3*− *FLD4_4*". Thus it is confirmed that this pattern is a substring of another pattern. Then, we need to check the frequency difference. Since it is a substring of two patterns, a comparison is needed for both cases at the end node.

[2]There is also minimum length requirement when generating suffix. Suffix shorter than minimum length can never match any pattern.

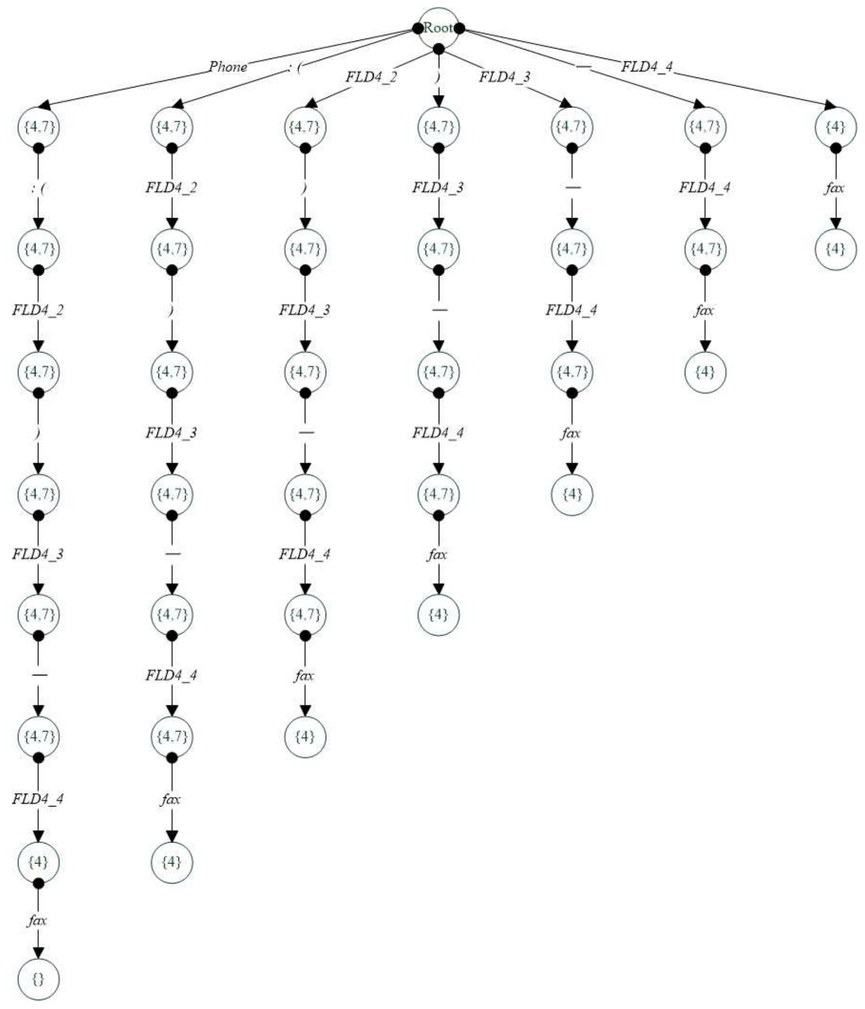


Figure 6.2: Example suffix tree

In addition, there is another problem that needs to be addressed, which is similar to the problem we encounter in the boundary detection. How close the frequencies should be for a substring pattern to be qualified as a redundant pattern.

The criterion we enforce here is as follows. Denote the frequency of the substring pattern that node i corresponds to as f_i and the set of frequencies stored in node i as $\{f_{i1}, f_{i2}, \ldots, f_{in}\}$, which correspond to the frequencies of the superstring patterns. Given a threshold t, if there exists an f_{ij} in the set satisfying $f_{ij}/f_i > t$, the substring pattern can be deemed redundant. Next, we experiment with different values of t.

6.3 Experiment

Similar to our experiment in Section 6.1.3, different values for the threshold are tried and the experiment is conducted on our all four different datasets.

In this step, the goal is to remove patterns that are part of other patterns but not general enough. Frequency is just a measure of generality and the annotators need to decide if a detected pattern p is general or not compared to the superstring patterns $\{p_1, p_2, \ldots, p_n\}$. However, among patterns $\{p_1, p_2, \ldots, p_n\}$, some may have frequency close to pattern p while some may not. To simplify the task, the annotators only compare pattern p with pattern $p_i\,(1 \leq i \leq n)$ such that the difference in frequency between the patterns is within the threshold. And to determine whether pattern p is more general, the annotators detect whether pattern p_i and substring pattern p express the same meaning and if it is safe to discard pattern p. In the case that both substring pattern p and pattern p_i provide no concrete semantic meaning, it is safe to discard a detected pattern p and it is marked as a correct detection. But when the annotators are not certain about the removal of a substring pattern, it is marked as incorrect.

For dataset Homepage and Birth, we take a 20% random sample of the removed patterns. For dataset Olympic and NBA, due to their sizes, the sample size was set to 5% of the removed patterns. The result is shown in Table 6.3.

Filter Threshold	80%			90%		
	True	Total	Precision	True	Total	Precision
Homepage	19	29	65.52%	24	34	70.56%
Birth	21	35	60%	23	33	69.70%
Olympic	26	37	70.27%	24	35	68.57%
NBA	42	61	68.85%	41	60	68.33%

Table 6.3: Precision of detecting redundant patterns as the threshold on frequency closeness varies

As the table shows, a fairly high threshold is needed to ensure all patterns removed are genuinely shorter duplicate of another pattern. For the datasets

Homepage and Birth, when the threshold drops, the precision decreases to approximately 60%.

However, for the Olympic and the NBA datasets, the decline of the threshold seems to have little effect on the precision. This may be because all the documents in Olympic and NBA datasets come from the same source, Wikipedia. Thus among these documents, there can be some phrases describing the structure and formatting. Since they appear almost in every document and in the same format, they are most likely to be detected in this step. In our experiments, the annotators ran across this type of substring patterns the most. As a result, even when the threshold drops, these patterns still account for a large portion of the detected patterns. In turn, the precision will not change much. On the other hand, this phenomenon shows that Substring Filtering works for homogeneous documents and such patterns are detected.

As for selecting a selecting threshold, the performance on the datasets Olympic and NBA should not be taken into account since they are not affected much by the threshold. Based on the performance on the other two datasets, the threshold should be set high, in the range of 90%.

Chapter 7

Pattern Ranking

After adding generic patterns and removing redundant ones, we expect the generality of the patterns to increase and the redundancy to drop. However, there still can be a large number of patterns and it would be better if we can order these patterns according to some measure of quality. This will make it more convenient for users, for example, if they want to adopt top n patterns instead of searching through a large set.

To sort the resulting patterns, we need some measures of quality or usefulness, and our discussions in the previous chapters provide us some clues.

7.1 Quality of Useful Patterns

In Chapter 5, it has been discussed that a pattern should be general. However, adoption of that measure of generality will lead to high ranks for all generic patterns, since non-generic patterns can only represent themselves and will be ranked down the list. This kind of bias against non-generic pattern is a problem. Another notion for generality is briefly discussed in Section 6.2.1. This notion focuses on the relationship between generality and the frequency of mentions. The logic here is if a pattern is general, it must be mentioned more often in the relevant documents. Also as mentioned in Section 3.3, the frequency of mentions is naturally a sign for popularity and representativeness. As a result, a large number of mentions of a pattern may indicate not only this pattern is representative, but also general.

The notion of specificity, as introduced in Chapter 5 can be applied here because it shows no preference on generic patterns or other patterns. Patterns with wildcard can still be more specific than some patterns without.

7.2 Ranking Method

We decided to order the patterns based on a mixture of their generality and specificity. Directly operating on the absolute values of the two measures is unsound due to differences in their scale. The frequency of mentions is an integer typically larger than 1 while specificity is a decimal between 0 and 1, which represents a probability.

To solve the problem of scale, candidate patterns are sorted first, based on the frequency of their mentions and specificity from the highest to the lowest. Thus, for a given pattern p, it has two ranking positions. Let $R_g(p)$ and $R_s(p)$ denote the ranks of pattern p based on respectively its generality and its specificity.

A simple mixture model is simply adding the two ranks, i.e. $R(p) = R_g(p) + R_s(p)$. In this way, the model treats both the specificity and the frequency of the same importance. But in reality, one quality may be more important than the other. An alternative ranking is to give the two quantities different weights, i.e.

$$\rho \cdot R_g(p) + (1 - \rho) \cdot R_s(p) \tag{7.1}$$

where ρ is the weight between 0 and 1 that is given to the generality measure. We refer to this mixture model as Simple Additive method (SA).

The second approach is inspired by the Mean Reciprocal Rank. Mean Reciprocal Rank (MRR) is a simple relevancy measure of performance of any process that produces a list of possible responses, for example, to match a query[1]. Consider a process that responds with a list of possible answers to a query. If the correct answer is at the ith position in the list, then the score of the answers given by the process is $1/i$, which is called the reciprocal rank. For a series of queries, the performance of the process is usually measured by the mean score for the queries.

[1] `https://en.wikipedia.org/wiki/Mean_reciprocal_rank`

In our case, we treat the reciprocal rank given to a pattern in a ranking method as an importance score. If a pattern ranks 1st according to the generality, the score the pattern gets is 1. Ranking 2nd means the score of 1/2. Consequently, for each pattern, it acquires two importance scores assigned by the aforementioned ranking methods. Now we can integrate these two scores in an additive model similar to Formulation 7.1, i.e.

$$\rho \cdot \frac{1}{R_g(p)} + (1-\rho) \cdot \frac{1}{R_s(p)} \tag{7.2}$$

where ρ is, as in Equation 7.1, the weight assigned to generality. We refer to this ranking as Additive Reciprocal Rank (ARR).

The difference between these two methods (i.e. Eq. 7.1 and 7.2) is that reciprocal rank emphasize the difference in ranking positions on top but neglects the difference in positions further down the list. In a reciprocal ranking, the score difference between ranking 1st and 2nd is far more larger than ranking 10th and 11th. On the other hand, simple additive model treats them the same since the distance between two ranking position are all 1. The next section reports our experiments on evaluating the two ranking methods.

Another difference is the ranking order. In SAM, a smaller score means a better pattern, while in SAM the larger

7.3 Experiment

In our experiment, we compare the performance of the two ranking methods in terms of the quality of top 5, 10, 20 and 30 results. We also try different mixture ratios to determine which aspect (i.e. generality and specificity) should be given more weight.

our experiments in previous steps aimed at detecting the redundancy, while here the annotators check whether a given pattern describes the semantics behind the table. The annotators were given the following guideline:

> *A pattern is relevant if (1) it gives a context where the field is expected to appear in, (2) it is not vague (e.g. it gives the context for only part of the field) nor redundant (e.g. the pattern can be shortened).*

With this guideline, the annotators are able to mark the holistic quality of the patterns, with each pattern marked as good or not. The percentage of good patterns are computed respectively for top 5, 10, 20 and 30 patterns.

From our experiments reported earlier in this thesis, it can be seen that documents from the same site can generate an overwhelmingly large number of patterns describing the format. For that reason and due to the evaluation cost, we decided not to use the NBA and the Olympic datasets in this experiment. Thus the pattern ranking methods are tested on the Homepage and Birth datasets. The result is shown in Tables 7.1 and 7.2. As a baseline, the last column in our tables shows the performance of ordering the patterns solely based on the number of mentions.

Top N	Additive Reciprocal Rank				Simple Additive Model				Frequency
	0.4	0.5	0.7	0.9	0.4	0.5	0.7	0.9	
5	80%	80%	80%	80%	80%	80%	80%	80%	80%
10	80%	80%	80%	80%	80%	80%	90%	80%	70%
20	75%	80%	80%	70%	55%	60%	75%	70%	60%
30	56.67%	56.67%	63.33%	63.33%	53.33%	56.67%	56.67%	63.33%	56.67%

Table 7.1: Experiments result for Homepage dataset of different weight ρ for top N patterns

Top N	Additive Reciprocal Rank				Simple Additive Model				Frequency
	0.4	0.5	0.7	0.9	0.4	0.5	0.7	0.9	
5	60%	60%	60%	80%	60%	40%	60%	80%	80%
10	60%	60%	60%	70%	40%	40%	70%	70%	70%
20	55%	50%	60%	60%	45%	50%	60%	55%	60%
30	46.67%	46.67%	46.67%	46.67%	46.67%	68.33%	46.67%	46.67%	50%

Table 7.2: Experiments result for Birth dataset of different weight ρ for top N patterns

In the table for the Homepage dataset, the performance difference between methods for different weights seems little. However, for the Birth dataset, the ranking that gives more weight to occurrence seem to perform better, which indicates that the frequency may be more important than specificity. This is reasonable because the frequency represents the popularity, and that may be primary factor to be taken into consideration when selecting good patterns. In our experiment, the performance difference seems to be negligible between the two methods. As we looked into the result, we found that the sets of patterns ranked top by the two methods are similar, with small differences in the orderings. One possible reason is that the patterns may be truly meaningful, and the two methods both identify these patterns and rank them top on the list.

In order to evaluate the performance of our ranking functions in placing "good" patterns on top, we used the top N patterns returned by each method (for different values of N) to find more instances of the rows. For the Homepage dataset, we randomly chose a half of its rows and used it as the training set to construct our pattern sets. The other half was used as the evaluation set. In other words, the training set and their corresponding documents were the input to our system. The top N patterns in the output were used to extract instances from the documents of the evaluation set and these instances were examined to see if they were truly the indicated rows and columns (or part of columns) of the table. The precision shown in Table 7.3 is the average precision over three runs, with each run taking different sets for training and testing. As shown, the baseline method that orders the patterns based on frequency

Top N	Additive Reciprocal Rank				Simple Additive Model				Frequency
	0.4	0.5	0.7	0.9	0.4	0.5	0.7	0.9	
5	51.52%	54.07%	53.40%	48.40%	62.67%	62.67%	68.29%	64.58%	38.94%
10	47.16%	45.00%	44.93%	45.97%	53.35%	53.35%	53.35%	52.45%	31.02%
20	40.55%	41.40%	40.39%	40.39%	41.98%	41.98%	41.98%	41.55%	16.71%

Table 7.3: Precision of extraction of top N patterns for different weight ρ

does not do well; this method returns general patterns that can extract many matches but few of them are correct. Of the two methods Additive Reciprocal Rank (ARR) and Simple Additive Model (SAM), SAM returns patterns with higher precision. To better understand this, consider a pattern ranked 1st and 99th respectively according to its specificity and frequency; the score this pattern gets using ARR is still larger than most patterns since the score is larger than one. But this is not a problem in SAM since the positions are treated equally. And this results in the precision difference between these two methods.

Chapter 8

Conclusions

In this thesis, we investigate in the problem of annotating web tables using surface text patterns. Our work shows that text patterns can be extracted to represent the semantics of a table, in terms of relationships between the columns, and that we can improve the pattern quality through increasing the generality and reducing the redundancy. It is observed that we need to balance between the generality and the specificity to avoid a possible over-generalization. Also, we have filtered patterns that are redundantly long and short. The experiments show these processings have a positive effect. Moreover, we construct a ranking method considering both frequency and specificity. Our experiments indicate that the pattern frequency should be given more weight.

8.1 Future Work

First, external resources such as Part of Speech tags and other forms of types may be used to improve the quality of our text patterns to make them more informative.

Second, text patterns may be used to associate a column with a predefined types from a database. As introduced in related work, linking table columns with predefined types may allow more structured operations on a table. A particular type can have a distinct set of text patterns describing it. Columns of a table can be associated with such types by checking the similarity between patterns describing them.

Third, with these text pattern as annotation, many operations can be available for the table. For instance, text patterns can be used for table integration. If two columns from different tables share a similar set of text patterns, they may represent the same type of objects. Another example for such application is entity resolution for table cells. And the most straightforward application is table enlargement. Text patterns can be utilized to locate additional entities and relationships that may not be mentioned in the table, and to enlarge the content of a table.

Bibliography

[1] Marco D. Adelfio and Hanan Samet. Schema extraction for tabular data on the web. *Proc. VLDB Endow.*, 6(6):421–432, apr 2013. ISSN 2150-8097. doi: 10.14778/2536336.2536343. URL `http://dx.doi.org/10.14778/2536336.2536343`.

[2] Danushka Tarupathi Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. Relational duality: Unsupervised extraction of semantic relations between entities on the web. In *Proc. of the 19th International Conference on World Wide Web*, WWW '10, pages 151–160, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772707. URL `http://doi.acm.org/10.1145/1772690.1772707`.

[3] Sergey Brin. Extracting patterns and relations from the world wide web. In *Selected Papers from the International Workshop on The World Wide Web and Databases*, WebDB '98, pages 172–183, London, UK, UK, 1999. Springer-Verlag. ISBN 3-540-65890-4. URL `http://dl.acm.org/citation.cfm?id=646543.696220`.

[4] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: Exploring the power of tables on the web. *Proc. VLDB Endow.*, 1(1):538–549, August 2008. ISSN 2150-8097. doi: 10.14778/1453856.1453916. URL `http://dx.doi.org/10.14778/1453856.1453916`.

[5] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL `http://doi.acm.org/10.1145/1327452.1327492`.

[6] Mark A Greenwood and Robert Gaizauskas. Using a named entity tagger to generalise surface matching text patterns for question answering. In *Proc. of the Workshop on Natural Language Processing for Question Answering (EACL03)*, pages 29–34. Citeseer, 2003.

[7] Eduard Hovy, Ulf Hermjakob, and Deepak Ravichandran. A question/answer typology with surface text patterns. In *Proc. of the Second International Conference on Human Language Technology Research*, HLT '02, pages 247–251, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org.login.ezproxy.library.ualberta.ca/citation.cfm?id=1289189.1289206`.

[8] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proc. of VLDB Endow.*, 3(1-2):1338–1347, September 2010. ISSN 2150-8097.

doi: 10.14778/1920841.1921005. URL `http://dx.doi.org/10.14778/1920841.1921005`.

[9] Rada Mihalcea and Andras Csomai. Wikify!: Linking documents to encyclopedic knowledge. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 233–242, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-803-9. doi: 10.1145/1321440.1321475. URL `http://doi.acm.org/10.1145/1321440.1321475`.

[10] Varish Mulwad, Tim Finin, Zareen Syed, and Anupam Joshi. Using linked data to interpret tables. In *In Proc. 1st Int. Workshop on Consuming Linked Data*, 2010.

[11] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 0215–0215. IEEE Computer Society, 2001.

[12] Deepak Ravichandran and Eduard Hovy. Learning surface text patterns for a question answering system. In *Proc. of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 41–47, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073092. URL `http://dx.doi.org.login.ezproxy.library.ualberta.ca/10.3115/1073083.1073092`.

[13] Zareen Syed, Tim Finin, Varish Mulwad, and Anupam Joshi. A.: Exploiting a web of semantic data for interpreting tables. In *Proc. of the Second Web Science Conference*, 2010.

[14] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *Proc. of VLDB Endow.*, 4(9):528–538, June 2011. ISSN 2150-8097. doi: 10.14778/2002938.2002939. URL `http://dx.doi.org/10.14778/2002938.2002939`.

Appendix A

User Settings

Table A.1: User setting file explanation

inputlistfile	Indicates the path of the input table file
inputwebpagesdirectory	Indicates the path of the directory containing all the input documents
outputpatternfile	Indicates the path of the output file containing the text patterns
keycolumnnum	Indicates the column in which entries are unique across all rows and distinguishes one row from all other
filterthreshold	Indicate the minimum occurrence that result patterns must have
removehtmltag	Indicate whether to remove HTML tags from textual context or not
substringfilter	Indicate whether to remove patterns that are sub-string of another pattern with close occurrence
numofgaps	Indicate whether removing patterns that are sub-string of other patterns with close occurrence
windowsize	Indicate maximum number and minimum number of words a text pattern can contain

Appendix B

Experiment Tables and Result Patterns

J Nelson Amaral	Professor	Athabasca Hall 352	1-780-492-5411	amaral @cs.ualberta.ca	amaral
Denilson Barbosa	Associate Professor	Athabasca Hall 451	1-780-492-9920	denilson @cs.ualberta.ca	denilson
Anup Basu	Professor	Athabasca Hall 402	1-780-492-3330	anup @cs.ualberta.ca	anup
Walter Bischof	Professor	Athabasca Hall 419	1-780-492-3114	wfb @cs.ualberta.ca	wfb
Pierre Boulanger	Professor	Athabasca Hall 411	1-780-492-3031	pierreb @cs.ualberta.ca	pierreb
Michae Bowling	Professor	Athabasca Hall 339	1-780-492-1766	bowling @cs.ualberta.ca	bowling
Vadim Bulitko	Associate Professor	Athabasca Hall 338	1-780-492-3854	bulitko @cs.ualberta.ca	bulitko
Michael Buro	Professor	Athabasca Hall 337	1-780-492-1763	mburo @cs.ualberta.ca	mburo
Renee Elio	Associate Vice President	Athabasca Hall 320	1-780-492-1734	ree @cs.ualberta.ca	ree
Ehab Elmallah	Professor	Athabasca Hall 309	1-780-492-9917	ehab @cs.ualberta.ca	ehab
Zac Friggstad	Assistant Professor	Athabasca Hall 306	1-780-492-2983	zacharyf @cs.ualberta.ca	zacharyf
Randy Goebel	President and CEO	Athabasca Hall 347	1-780-492-2683	goebel @cs.ualberta.ca	goebel
Russ Greiner	Professor	Athabasca Hall 359	1-780-492-5461	greiner @cs.ualberta.ca	greiner

Janelle Harms	Professor	Athabasca Hall 324	1-780-492-2763	harms @cs.ualberta.ca	harms
Ryan Hayward	Professor and Outreach	Athabasca Hall 301	1-780-492-3895	hayward @cs.ualberta.ca	hayward
Abram Hindle	Assistant Professor	Athabasca Hall 447	1-780-492-2285	hindle1 @cs.ualberta.ca	hindle1
Robert Holte	Professor	Athabasca Hall 349	1-780-492-3105	holte @cs.ualberta.ca	holte
Jim Hoover	Chair	Athabasca Hall 308	1-780-492-5290	hoover @cs.ualberta.ca	hoover
Martin Jagersand	Associate Professor	Athabasca Hall 401	1-780-492-5496	jag @cs.ualberta.ca	jag
Greg Kondrak	Associate Professor	Athabasca Hall 351	1-780-492-1779	kondrak @cs.ualberta.ca	kondrak
Guohui Lin	Associate Professor	Athabasca Hall 353	1-780-492-3737	ghlin @cs.ualberta.ca	ghlin
Paul Lu	Professor	Athabasca Hall 340	1-780-492-7760	paullu @cs.ualberta.ca	paullu
Mike MacGregor	Professor	Athabasca Hall 319	1-780-492-7434	macg @cs.ualberta.ca	macg
Martin Mueller	Professor	Athabasca Hall 345	1-780-492-3703	mmueller @cs.ualberta.ca	mmueller
Mario Nascimento	Professor	Athabasca Hall 434	1-780-492-5678	mn @cs.ualberta.ca	mn
Ioanis Nikolaidis	Professor	Athabasca Hall 322	1-780-492-5757	yannis @cs.ualberta.ca	yannis
Davood Rafiei	Associate Professor	Athabasca Hall 436	1-780-492-2374	drafiei @cs.ualberta.ca	drafiei
Nilanjan Ray	Associate Professor	Athabasca Hall 406	1-780-492-3010	nray1 @cs.ualberta.ca	nray1
Mohammad Salavatipour	Associate Professor	Athabasca Hall 303	1-780-492-1759	mreza @cs.ualberta.ca	mreza
Joerg Sander	Associate Professor	Athabasca Hall 432	1-780-492-5084	joerg @cs.ualberta.ca	joerg
Jonathan Schaeffer	Dean of Science	Athabasca Hall 415	1-780-492-5471	jonathan @cs.ualberta.ca	jonathan
Christian Schlegel	Adjunct Professor	Athabasca Hall 323	1-780-492-3255	schlegel @cs.ualberta.ca	schlegel
Dale Schuurmans	Professor	Athabasca Hall 409	1-780-492-4806	dale @cs.ualberta.ca	dale

Paul Sorenson	Professor Emeritus	Athabasca Hall 317	1-780-492-1564	sorenson @cs.ualberta.ca	sorenson
Lorna Stewart	Professor	Athabasca Hall 302	1-780-492-2982	stewart @cs.ualberta.ca	stewart
Eleni Stroulia	Professor	Athabasca Hall 452	1-780-492-3520	stroulia @cs.ualberta.ca	stroulia
Rich Sutton	Professor	Athabasca Hall 313	1-780-492-4584	rsutton @cs.ualberta.ca	rsutton
Duane Szafron	Vice Provost and Associate Vice President	Athabasca Hall 358	1-780-492-5468	duane @cs.ualberta.ca	duane
Csaba Szepesvari	Associate Professor	Athabasca Hall 311	1-780-492-8581	szepesva @cs.ualberta.ca	szepesva
David Wishart	Professor	Athabasca Hall 341	1-780-492-0383	dwishart @cs.ualberta.ca	dwishart
Ken Wong	Associate Professor	Athabasca Hall 305	1-780-492-5202	kenw @cs.ualberta.ca	kenw
Herb Yang	Professor	Athabasca Hall 413	1-780-492-3059	yang @cs.ualberta.ca	yang
Jia You	Professor	Athabasca Hall 354	1-780-492-5779	you @cs.ualberta.ca	you
Li-Yan Yuan	Professor	Athabasca Hall 356	1-780-492-7171	yuan @cs.ualberta.ca	yuan
Osmar R. Zaiane	Professor	Athabasca Hall 443	1-780-492-2860	zaiane @cs.ualberta.ca	zaiane
Hong Zhang	Professor	Athabasca Hall 407	1-780-492-7188	zhang @cs.ualberta.ca	zhang

Table B.1: Table in Homepage dataset

[[FLD4_2]]) [[FLD4_3]]-[[FLD4_4]]
[[FLD2_1]] department of computing science university of alberta
: ([[FLD4_2]]) [[FLD4_3]]-[[FLD4_4]] fax: ([[FLD4_2]]) [[FLD4_3]]-1071
: ([[FLD4_2]]) [[FLD4_3]]-[[FLD4_4]]
phone: ([[FLD4_2]]) [[FLD4_3]]-[[FLD4_4]]
phone: ([[FLD4_2]]) [[FLD4_3]]-[[FLD4_4]] fax: ([[FLD4_2]]) [[FLD4_3]]-1071
fax: ([[FLD4_2]]) [[FLD4_3]]-1071

[[FLD2_1]] department of computing science
tel: ([[FLD4_2]]) [[FLD4_3]]-[[FLD4_4]] fax: ([[FLD4_2]]) [[FLD4_3]]-1071
*[[FLD4_2]]***[[FLD4_3]]-1071*
*[[FLD4_2]]***[[FLD4_3]]-[[FLD4_4]] fax*

Table B.2: Top patterns for Homepage dataset

Mozart	1756	1791	Salzburg
Gandhi	1869	1948	Porbandar
Newton	1642	1727	Woolsthorpe
Einstein	1879	1955	Ulm
Washington	1732	1799	Westmoreland
Churchill	1874	1965	Blenheim Palace
Roosevelt	1882	1945	Hyde Park
Patton	1885	1945	San Gabriel
Hitler	1889	1945	Braunau am Inn
Armstrong	1930	2012	Wapakoneta
Turing	1912	1954	Paddington
Gates	1955		Seattle
Kennedy	1917	1963	Brookline
Edison	1847	1931	Milan
Bell	1847	1922	Edinburgh
Tesla	1856	1943	Smiljan
Bohr	1922	2009	Copenhagen

Table B.3: Table in Birth dataset

([[FLD2_1]]***[[FLD3_1]])
born in [[FLD4_1]]
persondata name [[FLD1_1]]
[[FLD1_1]] ([[FLD2_1]]***[[FLD3_1]]
([[FLD2_1]]***[[FLD3_1]]
[[FLD2_1]] place of birth [[FLD4_1]]
jump to: navigation, search "[[FLD1_1]]" redirects
[[FLD1_1]]" redirects here. for other uses, see
was born in [[FLD4_1]]
place of birth [[FLD4_1]]
search "[[FLD1_1]]" redirects here. for other uses
[[FLD1_1]] was born

Table B.4: Top patterns for Birth dataset

Athens	Greece	Europe	1896
Paris	France	Europe	1900
St. Louis	United States	North America	1904
Athens	Greece	Europe	1906
London	United Kingdom	Europe	1908
Stockholm	Sweden	Europe	1912
Berlin	Germany	Europe	1916
Antwerp	Belgium	Europe	1920
Paris	France	Europe	1924
Amsterdam	Netherlands	Europe	1928
Los Angeles	United States	North America	1932
Berlin	Germany	Europe	1936
Tokyo	Japan	Asia	1940
London	United Kingdom	Europe	1944
London	United Kingdom	Europe	1948
Helsinki	Finland	Europe	1952
Melbourne	Australia	Oceania	1956
Rome	Italy	Europe	1960
Tokyo	Japan	Asia	1964
Mexico City	Mexico	North America	1968
Munich	West Germany	Europe	1972
Montreal	Canada	North America	1976
Moscow	Soviet Union	Europe	1980
Los Angeles	United States	North America	1984
Seoul	South Korea	Asia	1988
Barcelona	Spain	Europe	1992
Atlanta	United States	North America	1996
Sydney	Australia	Oceania	2000
Athens	Greece	Europe	2004
Beijing	China	Asia	2008
London	United Kingdom	Europe	2012

Table B.5: Table in Olympic dataset

medal count[edit] main article: [[FLD4_1]] summer olympics
edit] main article: [[FLD4_1]] summer olympics medal table
commons has media related to [[FLD4_1]] summer olympics

references[edit] "[[FLD1_1]] [[FLD4_1]]". olympic.org. international olympic
the [[FLD4_1]] games
[[FLD4_1]] summer olympics -
] medal count[edit] main article: [[FLD4_1]] summer
olympiad ([[FLD4_1]]) succeededby
venues[edit] main article: venues of the [[FLD4_1]]
[[FLD1_1]] 1900-39 the blitz 1945-2000 21st
count[edit] main article: [[FLD4_1]] summer olympics medal
*the [[FLD4_1]]*** olympics*
performance-enhancing drugs in the olympic games -[[FLD4_1]]

Table B.6: Top patterns for Olympic dataset

Boston Celtics	Boston	MA	TD Garden	1946	
Brooklyn Nets	New York City	NY	Barclays Center	1967	1976
New York Knicks	New York City	NY	Madison Square Garden	1946	
Philadelphia 76ers	Philadelphia	PA	Wells Fargo Center	1946	1949
Toronto Raptors	Toronto	ON	Air Canada Centre	1995	
Chicago Bulls	Chicago	IL	United Center	1966	
Cleveland Cavaliers	Cleveland	OH	Quicken Loans Arena	1970	
Detroit Pistons	Auburn Hills	MI	The Palace of Auburn Hills	1941	1948
Indiana Pacers	Indianapolis	IN	Bankers Life Fieldhouse	1967	1976
Milwaukee Bucks	Milwaukee	WI	BMO Harris Bradley Center	1968	
Atlanta Hawks	Atlanta	GA	Philips Arena	1946	1949
Charlotte Hornets	Charlotte	NC	Time Warner Cable Arena	1988	
Miami Heat	Miami	FL	American Airlines Arena	1988	
Orlando Magic	Orlando	FL	Amway Center	1989	
Washington Wizards	Washington	D.C.	Verizon Center	1961	
Denver Nuggets	Denver	CO	Pepsi Center	1967	1976
Minnesota Timberwolves	Minneapolis	MN	Target Center	1989	
Oklahoma City Thunder	Oklahoma City	OK	Chesapeake Energy Arena	1967	
Portland Trail Blazers	Portland	OR	Moda Center	1970	
Utah Jazz	Salt Lake City	UT	EnergySolutions Arena	1974	
Golden State Warriors	Oakland	CA	Oracle Arena	1946	
Los Angeles Clippers	Los Angeles	CA	Staples Center	1970	
Los Angeles Lakers	Los Angeles	CA	Staples Center	1947	1948
Phoenix Suns	Phoenix	AZ	US Airways Center	1968	
Sacramento Kings	Sacramento	CA	Sleep Train Arena	1923	1948
Dallas Mavericks	Dallas	TX	American Airlines Center	1980	
Houston Rockets	Houston	TX	Toyota Center	1967	
Memphis Grizzlies	Memphis	TN	FedExForum	1995	
New Orleans Pelicans	New Orleans	LA	Smoothie King Center	2002	
San Antonio Spurs	San Antonio	TX	AT&T Center	1967	1976

Table B.7: Table in NBA dataset

main article: list of [[FLD1_1]] [[FLD1_2]]
main article: list of [[FLD2_1]] [[FLD1_2]]
records[edit] main article: list of [[FLD2_1]]
1969) pittsburgh pipers/condors ([[FLD5_1]]-1968; 1969-
[edit] main article: list of [[FLD2_1]] [[FLD1_2]]

[edit] main article: list of [[FLD1_1]] [[FLD1_2]]
commons has media related to [[FLD1_1]] [[FLD1_2]]
coaches[edit] main article: list of [[FLD1_1]]
based in [[FLD1_1]]
founded in [[FLD5]] based in [[FLD2_1]]
records[edit] main article: list of [[FLD1_1]]
commons has media related to [[FLD2_1]] [[FLD1_2]]
the [[FLD1_1]] [[FLD1_2]]
pittsburgh pipers/condors ([[FLD5]]-1968; 1969-1972)
*basketball nba***[[FLD1_1]] [[FLD1_2]]*
" categories: [[FLD1_1]]
-1969) pittsburgh pipers/condors ([[FLD5_1]]-1968; 1969

Table B.8: Top patterns for NBA dataset