



All Theses and Dissertations

2007-08-02

Text Identification by Example

Daniel Joseph Preece

Brigham Young University - Provo

Follow this and additional works at: <https://scholarsarchive.byu.edu/etd>



Part of the [Computer Sciences Commons](#)

BYU ScholarsArchive Citation

Preece, Daniel Joseph, "Text Identification by Example" (2007). *All Theses and Dissertations*. 1184.
<https://scholarsarchive.byu.edu/etd/1184>

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

TEXT IDENTIFICATION BY EXAMPLE

By

Daniel Joseph Preece

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree
of

Master of Science

Department of Computer Science
Brigham Young University
December 2007

Copyright © 2007 Daniel Preece

All Rights Reserved

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Daniel Joseph Preece

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date

Dan Olsen, Chair

Date

Parris Egbert

Date

Daniel Zappala

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Daniel Joseph Preece in its final form and have found the (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Dan Olsen
Chair, Graduate Committee

Accepted for the Department

Parris K. Egbert
Graduate Coordinator

Accepted for the College

Thomas W. Sederberg
Associate Dean, College of Physical and
Mathematical Sciences

ABSTRACT
Text Identification By Example

Daniel Joseph Preece

Department of Computer Science

Master of Science

The World-Wide Web contains a lot of information and reading through the web pages to collect this information is tedious, time consuming and error prone. Users need an automated solution for extracting or highlighting the data that they are interested in. Building a regular expression to match the text they are interested in will automate the process, but regular expressions are hard to create and certainly are not feasible for non-programmers to construct. Text Identification by Example (TIBE) makes it easier for end-users to harvest information from the web and other text documents. With TIBE, training text classifiers from user-selected positive and negative examples replaces the hand-writing of regular expressions. The text classifiers can then be used to extract or highlight text on web pages.

ACKNOWLEDGMENTS

Thanks to my wife Alicia who has put up with me for the last 4 years and my two sons, Connor and Haden, who were both born after I started working on this thesis.

TABLE OF CONTENTS

Chapter 1 – Introduction	1
1.1 TIBE and Automated Data Extraction.....	3
1.2 Applications of TIBE.....	5
1.3 Delimitations of the Thesis	7
1.4 Training Scenario.....	7
1.5 Training TIBE	9
1.6 Measuring TIBE	10
Chapter 2 – Prior Work.....	11
2.1 Concept Learning	11
2.2 TELS: Learning Text Editing Tasks from Examples	15
2.3 LAPIS and Multiple Selections in Smart Text Editing.....	16
2.4 Roadrunner.....	18
2.5 Conclusions	19
Chapter 3 – Overall Approach	20
Chapter 4 – The TIBE User-Interface.....	23
4.1 Training Example	24
4.2 Implementation Details.....	32
Chapter 5 – TIBE Training Engine	34
5.1 TIBE Patterns.....	35
5.2 Converting a Pattern into a Classifier.....	39
5.3 TIBE Training Algorithm.....	41
5.4 Pattern Generalization.....	43
5.4.1 Deriving the minimal pattern.....	46
5.4.2 Making the general boundary more specific	47
5.4.3 TIBE Training Rules.....	48
5.5 TIBE Trainer Scenarios	53
5.6 Training Dates on a Genealogy site.	53
5.6.1 Generalizing a “Date” Positive Training Example.....	54
5.6.2 Matching more than one positive training example.....	57
5.6.3 Matching a Negative Training Example.....	57
5.7 Training List Price on Camera Pages.....	58
Chapter 6 – The TIBE Classifier.....	63
6.1 Classifying Text	63
6.2 Classifying/Recognizing Engine.....	65
Chapter 7 – Evaluation	66

7.1 Introduction.....	66
7.2 TIBE Analyzer	66
7.3 Computer Generated Webpage Content	67
7.3.1 Summary of Results.....	69
7.4 Human-Generated Webpage Content.....	71
7.4.1 Summary of Results.....	72
7.5 Conclusions	73
Chapter 8 – Summary	75
8.1 What was the problem?.....	75
8.2 The TIBE Solution.....	75
8.3 What remains to be done?	76
Bibliography	80

CHAPTER 1 – INTRODUCTION

The World-Wide Web contains a lot of information. There are many reasons to want to harvest that information. Genealogists want to extract information about the lives of the people they are researching. A stock broker wants to extract information about stocks that he might want to invest in. A user researching cameras wants to extract price and feature information about digital cameras. Reading through the web pages is tedious, time consuming and error prone. What these users really want is an automated solution to extracting or highlighting the data that they are interested in. Building a regular expression to match the text they are interested in will automate the process, but regular expressions are hard to create and certainly are not easy for non-programmers to construct (see Figure 1-1).

```
Regex for Email:
^[a-zA-Z0-9_\-\.]+@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.))|([a-zA-Z0-9\-\.\_]+))([a-zA-Z]{2,4}|[0-9]{1,3})\(\)?

Date: DD MMM YYYY
^((31(?! (FEB|APR|JUN|SEP|NOV)))|((30|29)(?! FEB))|(29(?! FEB
B ((1[6-9]|[2-9]\d)(0[48]|[2468][048]|[13579][26])|((16|[24
68][048]|[3579][26])00))))|(0?[1-9])1\d|2[0-8]) (JAN|FEB|M
AR|MAY|APR|JUL|JUN|AUG|OCT|SEP|NOV|DEC) ((1[6-9]|[2-9]\d)\d{
2}))$

Date Time: MM/DD/YYYY HH:MM:SS AM|PM
(?:n:^(?=\d)((?<month>(0?[13578])|1[02]|(0?[469]|11)(?! .31)|0
?2(?:.29)(?=.29)((1[6-9]|[2-9]\d)(0[48]|[2468][048]|[13579][
26])|(16|[2468][048]|[3579][26])00))|(?!.3[01]))(?<sep>[-./
]) (?<day>0?[1-9]|1[2]\d|3[01])\k<sep> (?<year>(1[6-9]|2[2-9]\d
)\d{2}) (?<time>(?:\x20\d)\x20{2}) (?<time>((0?[1-9]|1[012]) (: [0-5]
\d) (0,2) (?i:\x20[AP]M)) | ([01]\d|2[0-3]) (: [0-5]\d) {1,2}) ?$)
```

Figure 1-1: Various hard-coded regular expressions.

The goal of Text Identification by Example (TIBE) is to make it easier for end-users to harvest information from the web and other text documents. Training text classifiers from user-selected examples replaces hard-coded regular expressions like Figure 1-1 with a user interface where the text identification can be built through user examples as in Figure 1-2.

List Price: \$14.95
Price: \$10.17 & eligible for
FREE Super Saver
Shipping on orders over
\$25. [Details](#)
You Save: \$4.78 (32%)

Figure 1-2: Trained dollar amounts are highlighted.

This process of training a text classifier is intuitive and simple for end-users. As text classifiers are trained within the user interface, instant classifier feedback is provided, allowing the user to quickly and accurately train the classifier. Classifiers are trained by the user selecting positive and negative examples. These classifiers are then used to identify other instances of the text class the user is training. As other instances of the text class are identified by the classifier, the user adds additional text examples that were missed by the classifier and marks as negative any text examples that were identified in error by the classifier. The trainer uses the additional positive and negative examples to fine-tune the classifier until the user is satisfied that it is classifying the text properly.

Using TIBE to train a text classifier simplifies the process of automating the extraction of text from data rich web pages compared to manually browsing web pages or manually creating hard-coded regular expressions. In *A conceptual-modeling approach to extracting data from the web* [Emb98B], a document is considered data rich if it has a number of identifiable constants such as dates, names, account numbers, ID numbers, part numbers, times, currency, values and so forth. For these data rich documents, browsing is not suitable for locating items to extract because following links is tedious, it is easy to get lost, and browsing is not cost-effective as users must read the whole document to find the desired data [Emb98B]. Automating the extraction by creating hard-coded regular expressions is difficult. The regular expression language is commonly

called the world's most popular "write-only" language because it is so hard to read. TIBE solves both of these problems.

1.1 TIBE and Automated Data Extraction

There has been considerable research into building other tools that automate the extraction of meaningful information from documents on the Web. Conceptual-models and ontologies [Emb98A], [Emb98B], [Emb99], [Emb04A], [Emb04B] are two methods used to extract data and to relate the data together. In each of the methods of extraction, there is a step where regular expressions must be hand-coded to identify the individual elements of their models. In Figure 1-3, this is the "Constant/Keyword Recognizer" step in the data extraction and structuring process. In Ontology-Based Extraction [Emb98A], Embley et al. state that their process uses a domain expert to tune their system. A domain expert is an individual that is familiar with hard-coding regular expressions to act as the Constant/Keyword Recognizers for the domain of documents that they are extracting data from. Because the output of TIBE is a classifier (i.e. recognizer), TIBE could be used to build those recognizers to extract text from data-rich documents such as advertisements, movie reviews, weather reports, travel information, sports summaries, financial statements, obituaries, etc. that are fed into the process in Figure 1-3.

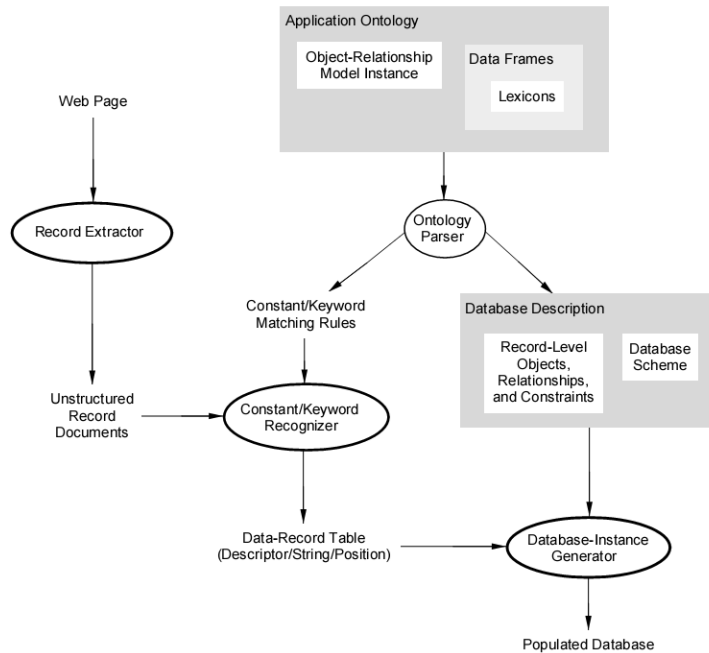
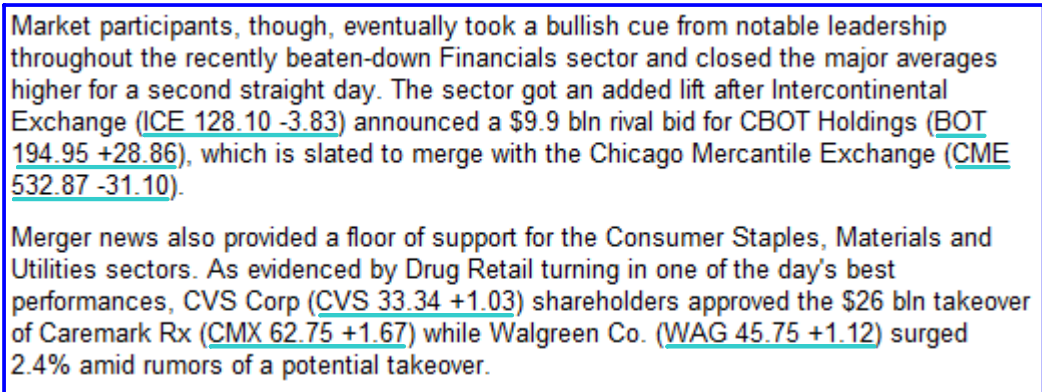


Figure 1-3: Model for Conceptual-Model-Based Data Extraction

To overcome the need to have the end user hard-code regular expressions, a fixed set of regular expressions can be pre-programmed by the regular expression experts, but there are infinite types of data classifications and it is not feasible to pre-program every possible one. Adding an interface to be able to modify and add new regular expressions would allow for additional data extraction, but would still require a regular expression expert to program them. Again TIBE would replace these pre-coded regular expressions with a much more flexible and easier interface for a non-expert to use. The conceptual-models and ontologies have made it possible to extract data from web pages in a meaningful way and TIBE makes it more accessible to an end-user by replacing hand-coded regular-expressions created by domain experts with text classifiers created by end-users.

1.2 Applications of TIBE

Using TIBE to train text classifiers has a number of applications. For example, there might be a user that watches certain financial web sites on a regular basis for stocks that are mentioned in their articles. Reading through the page can be tedious and time consuming and this user does not have the time or the patience to scan the page for stock quotes or mentions of the stocks. With TIBE, he could train a text classifier to recognize the stock quotes and have them highlighted or have the paragraphs that contain mentions of the stocks extracted (see Figure 1-4).



Market participants, though, eventually took a bullish cue from notable leadership throughout the recently beaten-down Financials sector and closed the major averages higher for a second straight day. The sector got an added lift after Intercontinental Exchange (ICE 128.10 -3.83) announced a \$9.9 bln rival bid for CBOT Holdings (BOT 194.95 +28.86), which is slated to merge with the Chicago Mercantile Exchange (CME 532.87 -31.10).

Merger news also provided a floor of support for the Consumer Staples, Materials and Utilities sectors. As evidenced by Drug Retail turning in one of the day's best performances, CVS Corp (CVS 33.34 +1.03) shareholders approved the \$26 bln takeover of Caremark Rx (CMX 62.75 +1.67) while Walgreen Co. (WAG 45.75 +1.12) surged 2.4% amid rumors of a potential takeover.

Figure 1-4: Stock symbols that could be trained by TIBE.

An end user might want to research different cameras offered on Amazon. On the Amazon pages for cameras there are several pieces of information that he might want to collect in order to compare the cameras: List Price, Sale Price, Discount Percent, Mega-Pixels, Optical Zoom, LCD Display Size, ISO, Shipping Weight, Model Number, Sales Rank, etc. (see Figure 1-5). If he tries to collect this information himself, he must scan the page for the information and manually transfer it to a spreadsheet or a document to compare the cameras. He could create regular expressions to automate the extraction of this data and be able to process several pages in the time that it would have taken him to manually scan one, but he's a photographer, not a programmer and has no idea what regular

expressions are. With TIBE, within a few minutes he could have each of the data elements trained and then the classifiers could be used to extract the data just like the regular expressions would have.

Technical Details

- 5.0-megapixel CCD captures enough detail for photo-quality [13 x 17-inch prints](#)
- 4x optical zoom; 1.8-inch LCD display
- ISO 800 shooting; widescreen (16:9) stills recording
- DIGIC II and iSAPS for fast, precise results; VGA-quality movie mode with
- Powered by 2 AA-size batteries; stores images on SD memory cards (16 M

Product Details

Product Dimensions: [3.5 x 2.5 x 1.7 inches](#) ; 6.4 ounces

Shipping Weight: [3.00 pounds](#) ([View shipping rates and policies](#))

Shipping: Currently, item can be shipped only within the U.S.

ASIN: [B000EMU888](#)

Item model number: [1119B001](#)

Average Customer Review: [★★★★☆](#) based on 167 reviews. ([Write a review.](#))

Amazon.com Sales Rank: [#18](#) in Camera & Photo (See [Top Sellers in Camera & Photo](#))

Date first available at Amazon.com: [February 21, 2006](#)

Figure 1-5: Data items on an Amazon camera page.

Similarly, a genealogist researching names on web sites could also use TIBE. The genealogist could train several text classifiers for data that he is looking for. For example he could train dates and other genealogical related information and even specific instances of a general type like birth dates, death dates and marriage dates (See Figure 1-6).

1. **EVERETT** : Searching for half-siblings of my son. Born in Salt Lake City late 70's to one 15 year old girl and one 18 year old girl. My son has no siblings and I would love to locate you. Father was a bus driv ...
 Posted: 21-Jan-2007 | [\[View F](#)

2. **JOHNSON** : I am searching for and was born on 6/13/67 at Holy through The Children's Service S Mom was 23 years of age a ...
 Posted: 18-Jan-2007 | [\[View F](#)

Clayton Edward Bollschweiler 1907 ~ 2007 Clayton Edward Bollschweiler ended his life journey on January 21, 2007 at the home of his daughter, JoAnne Parkes, in Ivins, Utah. He is now happily reunited with his Lord, Jesus Christ, his dear wife, Lenore, and other loved ones waiting for him. Clayton was born on October 1, 1907, to Gustave and Alvina Breyhan Bollschweiler in Salt Lake City, Utah. He married Lenore Potter on March 23, 1927, in the Salt Lake City Temple, and she passed away on November 26, 1977. Clayton married Mabel Kemple on October 24, 1988 and they resided in St.

Figure 1-6: Genealogical information from genealogical or obituary sites.

1.3 Delimitations of the Thesis

TIBE's focus will be primarily the user interface, the algorithm for training text classifiers, and using the classifiers to match the appropriate data being classified. It will not address the conceptual-models, ontologies, or relating matched text to other elements on the page. After a classifier is trained by TIBE, it could be used in a highlighter or a text extractor or within the conceptual-model process, but these applications will not be the focus of this thesis, and neither are the steps to hook the TIBE classifier into these processes as that would be fairly straight-forward.

Because TIBE's interface is browser-based, there has been some work done to be able to train text within Windows Internet Explorer, but this is more of a proof-of-concept than a full implementation. TIBE's implementation in Windows Internet Explorer is good enough to demonstrate how the user interface works, but it is not part of this thesis to make it production quality.

1.4 Training Scenario

There are relatively few elements that are part of TIBE's user-interface. This helps make TIBE simple to use and quick to learn. The main function of the interface is to facilitate the training of text classifiers. There is also the ability within the interface to organize the text classifiers into different projects to support multiple types of web pages.

The process to train a classifier is straightforward. For example, suppose a user is on an Amazon book page and wants to train the book price. He chooses the "Amazon Book Pages" project and creates the "Price" text class. Because this class has not been trained yet no text in the browser is highlighted.

The user then highlights the book price with the mouse and presses the “Add Example” button. The TIBE trainer creates a classifier and highlights other text on the page that the classifier matched (See Figure 1-7).



Figure 1-7: The user adds the first training example.

There are several dollar amounts that were matched that are not the book price that is being trained and so the user highlights one of wrong matches and clicks the “Add Negative Example” button. This results in the trainer creating a new classifier and the classifier highlighting its matches (See Figure 1-8).

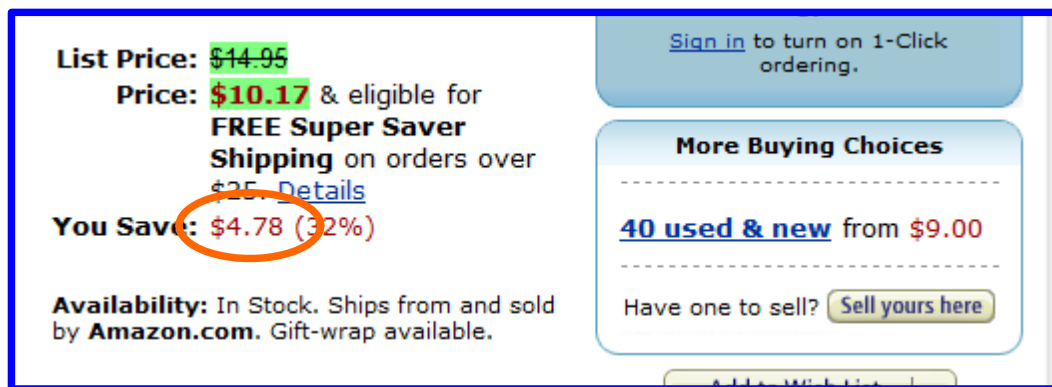


Figure 1-8: “\$4.78” is added as a negative training example.

After adding the negative training example, several of the dollar amounts that had been highlighted previously are no longer highlighted but there are still some dollar amounts that are erroneously being classified as a book price. Next the user selects one of them and clicks the “Add Negative Example” button again

which results in the trainer creating a new classifier and the classifier highlighting its matches again (See Figure 1-9).



Figure 1-9: "\$14.95" is added as a negative training example.

This new negative training example results in a couple of the dollar amounts being matched again by the classifier so the user selects one of them and adds it as another negative training example and the training and classification occur once again. This time though, the trainer creates a classifier that classifies the book price correctly and does not match any of the other dollar amounts as book prices (See Figure 1-10).

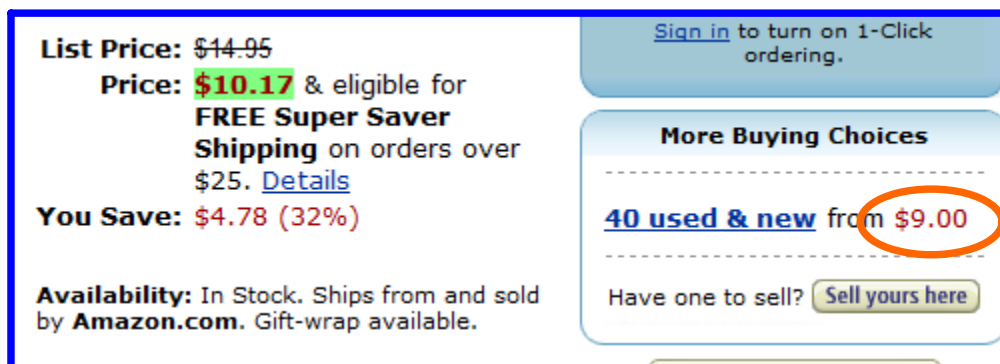


Figure 1-10: "\$9.00" is added as a negative training example.

1.5 Training TIBE

TIBE's training of a classifier is non-trivial. Generating minimal classifiers from text examples is generally computationally intractable [Wit93] and so

TIBE's generalization algorithm has novel optimizations to reduce the problem size and for finding minimal classifiers.

1.6 Measuring TIBE

The TIBE algorithm's precision and accuracy is measured by running it against a gold standard of pre-marked-up text. The pre-marked-up text consists of several different types of web sites with several text classes unique to each one. Some of the websites consist of computer generated content while others are human generated.

CHAPTER 2 – PRIOR WORK

TIBE's algorithms build upon concepts that have been around for a while. It is a concept learning algorithm that builds a classifier based on positive and negative examples. There are two groups of prior work that TIBE either builds upon or solves the problem in a different way. The first group of prior work deals with material that was used as building blocks to the algorithms in TIBE. The second group deals with material that is similar to TIBE and we explain how they are similar and what makes TIBE different.

2.1 Concept Learning

Concept learning is defined as acquiring the definition of a general category given a sample of positive and negative training examples of the category [Mit97]. TIBE uses a machine learning algorithm to generalize positive and negative training examples to train a classifier that will match other text fragments for that class and thus can be considered a concept learning algorithm. Concept learning algorithms have a hypothesis space that is the list of hypotheses that are consistent with the positive and negative training examples given. It is usually ordered from a most general boundary to a most specific boundary. In TIBE, the hypothesis space for a text class would be all patterns that match all of the positive examples and none of the negative examples.

From the positive and negative training examples, TIBE's learning algorithm produces the subset of the patterns in the hypothesis space that are on the most general boundary. These patterns are guaranteed consistent with the training examples, but the ultimate goal is to have a set of patterns that are consistent with the entire set of instances of the data. This means that the classifier created from the patterns should recognize all of the text that is part of the text class

being trained and none of the text that is not part of that text class. Since the only data available to the TIBE learning algorithm is the training data, it can at most guarantee that the output classifier from the patterns recognizes the training data. Lacking any further information, our assumption is that the best patterns regarding unknown instances of a text class are the patterns that best fit the observed training data. This is the fundamental assumption of inductive learning.

The inductive learning hypothesis. Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples. [Mit97]

For example, suppose the text class "Date" was being trained. Given one positive example "4/4/1977", there are several hypotheses that could be generated. The most specific hypothesis is "4/4/1977" and the most general is "<char>*". The most specific hypothesis matches the training examples but does not match other dates such as "7/2/1980" or "12/30/2005". However, the most general hypothesis matches other dates like "7/2/1980" or "1/1/2004" as well as all characters and so is a better classifier to match unobserved examples, but it matches other text like "Hospital" or "12:04 am"; therefore there must be an even better classifier. Adding additional negative training examples could make the most general hypothesis eventually converge to "<digit><symbol><digit><symbol><digit>+". Now the pattern matches the given positive training example and no negative training examples but it does not match some dates such as "12/30/2005". If "12/30/2005" is added as a positive example, then its most general hypothesis evaluates to "<digit>+<symbol><digit>+<symbol><digit>+" which not only matches all of the training examples, but also matches all dates with single or double digit months or days such as "7/2/1980", "12/4/1978", "12/30/2005" or "1/10/1955". With the additional positive

training example, the resulting classifier does a pretty good job matching other dates beside the given positive training examples.

Chapter 2 of *Machine Learning* [Mit97] reviews “Concept Learning” and several algorithms: Find-S, List-Then-Eliminate and Candidate-Elimination. The Find-S algorithm [Mit97] uses only the positive training examples and generates the most specific hypothesis that generalizes the training examples. The most specific hypothesis in Find-S is found by generalizing the most specific possible hypothesis (which is a hypothesis that matches nothing). Generalization occurs by replacing constraints in the hypothesis with more general constraints until the specific hypothesis matches all positive training examples. If Find-S were used with TIBE and “12/4/1978” and “12/30/2005” were the two positive examples, it would generate a specific hypothesis of “<digit><digit>/<digit><digit>/<digit><digit><digit><digit>” which would match the positive training examples and several other dates, but not dates like “4/4/1977” or “1/10/1955”. Therefore we would not want to use the Find-S algorithm for TIBE because we want to find the most general hypotheses for our classifiers (with rules on how far we will generalize a pattern) rather than the most specific ones, and in order to approximate the general hypothesis boundary, negative training examples might need to be used in addition to the positive ones.

The List-Then-Eliminate algorithm [Mit97] uses both the positive and negative training examples to get a list of hypotheses. It first initializes the hypothesis space to contain all possible hypotheses and then eliminates any hypothesis found inconsistent with any training example. Ideally this would reduce the hypothesis space to just one hypothesis that remains consistent with the training examples, but if the training examples are sparse, there could be more than one hypothesis in the resulting set. The subset of hypotheses that are

consistent with the training examples is called the *version space* with respect to the hypothesis space because it contains all plausible versions of the target concept. Although the List-Then-Eliminate algorithm is guaranteed to output all hypotheses consistent with the training data, it requires exhaustively enumerating all hypotheses in the hypothesis space. For TIBE this would not be trivial as the hypothesis space would be too large to compute for almost any text class.

A third concept learning algorithm is the Candidate-Elimination algorithm [Mit97]. This is a more compact representation of the List-Then-Eliminate algorithm. The version space is represented by its most general and least general members. These members form general and specific boundary sets that delimit the version space within the partially ordered hypothesis space. This algorithm starts with a general hypothesis that matches everything and a specific hypothesis that matches nothing and uses the positive and negative training examples to make the general boundary more specific and the specific boundary more general.

TIBE's training algorithm is most similar to the candidate-elimination algorithm except for a few notable differences. The TIBE learning algorithm only gets the general boundary instead of getting both the general and specific boundaries. It also has specific generalization rules that only generalize letters, numbers, whitespace, strings of the same pattern category and other text class patterns that are matched within the current pattern.

The Candidate-Elimination learning algorithm assumes that all positive examples are relatively the same, whereas in TIBE we have to accommodate training text classes that may have two items with totally different structures. For example, a date text class might be formatted as "9/23/2006" or "September

23, 2006" or even "23 Sept. 2006". In this case, the TIBE learning algorithm will detect that some of the positive training examples cannot be generalized with the other positive examples and will generalize them separately. With this algorithm, TIBE starts at the generic hypothesis boundary and uses other positive and negative examples to make the generic boundary more specific. For example, a positive example of the "Birth Date" class might be "Birth: 12/30/2005", which would be generalized in TIBE to "<digit>+<digit>+<digit>+". The "Birth: " part of the example would be generalized off as it is not needed to match the positive example. This pattern also happens to match "Death: 11/23/1999" because it has been generalized too far. If that death date is added as a negative training example, this would force the general hypothesis boundary to "rth:<whitespace><digit>+<digit>+<digit>+", which would match all the positive examples and none of the negative examples and would also match any other birth date that is preceded with "Birth: ".

2.2 TELS: Learning Text Editing Tasks from Examples

TELS generalizes the steps that a user makes when editing text into a program that automatically edits the text [Wit93]. This is often called "Programming by Demonstration". Similar to most other "Programming by Demonstration" systems, TIBE generalizes the positive and negative training examples that the user enters to create a classifier to match text. The problem that TELS is solving is different from the problem that TIBE is solving. TELS is dealing with generalizing editing tasks into an editing routine whereas TIBE is generalizing text examples into a classifier for recognizing the text.

The general algorithm for TELS could not be used for TIBE, but part of the TELS generalization involves generalizing character strings and some of these concepts were used within the TIBE generalization algorithm. In generalizing

character strings within the TELS algorithm, it seeks the smallest regular expression that satisfies the examples given. This can be found by enumerating all expressions from the smallest up, but unfortunately this is computationally intractable; therefore, TELS generalizes character strings using a heuristic that finds common subsequences. Individual characters are related by a class hierarchy with categories such as *letter*, *digit*, *punctuation*, *lower-case*, *non-alphanumeric*, and this relationship is extended to strings of consecutive characters from the same class. TIBE uses pattern categories similar to TELS's character categories. TIBE's pattern categories include letter, digit, whitespace and other TIBE trained classifiers as well as strings of consecutive pattern categories to aid in the generalization of training examples into a classifier.

2.3 LAPIS and Multiple Selections in Smart Text Editing

LAPIS is a prototype system that uses an approach to matching text called *lightweight structured text processing* [Mil99]. Within LAPIS, a user can use a pattern language called *text constraints* to aid in selecting text to be highlighted, extracted, sorted or edited. Text constraints describe text structure in high-level terms, with region relationships like *before*, *after*, *in* and *contains*. They also have the concept of domains like HTML, JAVA source code or English text which include parsers and sets of common patterns to aid in the pattern matching.

The LAPIS prototype system is extended to enable Multiple Selections in Smart Text Editing [Mil01], [Mil02]. This extension to the prototype enables a user to simultaneously edit multiple regions in a document. Whereas previous techniques for repetitive text editing generally separated records on the line boundary or something similar, Smart Text Editing allows the user to define the record set that the editing works on. After defining the records, the user makes a selection in one record using the mouse or keyboard, and the system makes an

equivalent selection in all other records. Subsequent editing operations – such as typed text, deletions, or cut-and-paste – affect all records simultaneously, as if the user had applied the operations to each record individually.

Smart Text Editing is trying to solve the same problem as other “Programming by Demonstration” (PBD) techniques such as TELS [Wit93], except the user’s demonstrations affect all records simultaneously. After the demonstration part of a transformation, the user can scan through the file and see how the other records were affected by the partial transformation. In TELS, each demonstration affects only a single example. In order to see what the inferred program will do to other examples, the user must run the program on other examples.

Smart Text Editing is similar to TIBE in a few ways. Both allow the user to give positive and negative training examples to train the generalizer and get instant feedback on the generalizer’s results. Both have a concept of domains. Smart Text Editing and LAPIS use domains to define the parser and other patterns that can aid in the training and pattern matching. TIBE has classifier domains, which contain text classes that are found in a set of related web pages. Similar to LAPIS, the text classes in a classifier domain can aid in the training and pattern matching of additional text classes in that domain.

Although the selection guessing portion of Smart Selections is similar to the training that TIBE does, it is trying to solve a different problem than TIBE and therefore, its implementation is not entirely suited to the problem that TIBE is trying to solve. Since Smart Selections is targeted towards editing rather than just text identification, the smart selections work upon records that the user defines (which might be lines, java methods, paragraphs, etc). This enables the user to enact the same editing command across all the records in the document.

It also enables Smart Selections to preprocess the records to aid in making the generalizing quicker. Since TIBE is not used for editing, it has no reason to divide sections of a page into records and so works on the page as a whole. It also does not need to preprocess the page. Another limiting factor of their algorithm is that if it cannot find a hypothesis that is consistent with all of the positive and negative text examples, it just does not produce a hypothesis and asks the user to adjust his examples. This might happen if they were training a date and had two positive examples of "10/31/2006" and "Oct. 31, 2006". The TIBE algorithm detects that these might require different hypotheses and generates them appropriately.

2.4 Roadrunner

Roadrunner [Cre01] is a tool for automating the extraction of information from web pages. It is similar to TIBE in that both aid in extracting useful information from web pages. Roadrunner analyzes a set of web pages and attempts to derive data fields on the web pages by comparing the web pages to each other. The data fields are not classified but rather organized into fields which have no specific class associated with them. Roadrunner also depends upon the pages being generated by a relational database and upon the pages' structure to be relatively the same. Roadrunner is also focused on automating the extraction of web pages (where a user would not even need to be involved in identifying the important fields on a web page).

Unlike Roadrunner, TIBE is able to classify the text on the web page. TIBE would require a user to identify text classes up front before the automated text extraction but this has the benefit of having classified fields extracted. TIBE also works on pages that were not necessarily generated by a relational database and which do not necessarily have relatively the same structure. While Roadrunner

analyzes several whole pages together, TIBE trains a single class over several pages using specific training examples.

2.5 Conclusions

The very nature of using “Programming by Demonstration” to teach a program how to edit text requires the algorithm to be able to learn patterns of text in order to detect where edits should occur in subsequent records. Both TELS and Smart Selections generalize text or selections in order to match similar text in other records. From TELS we got the idea of how to generalize individual characters and from Smart Selections we see the idea of instant feedback on selections. Both implementations ultimately are optimized for editing rather than text extraction and as a result have limitations that TIBE does not have.

The TIBE training algorithm most closely follows the Candidate-Elimination concept learning algorithm. This enables our algorithm to be able to approximate the general boundary of its version space without having to enumerate every possible combination of generalizations that a text example pattern could have.

The Roadrunner project is also focused on extracting data from web pages but it is focused more on automating the process rather than allowing users to interactively identify the data that they want to extract. It would not require a user to identify data to extract, but it also cannot classify the data it is extracting and depends on the web pages being generated by relational databases. TIBE classifies the data that it is identifying and it can work on all types of web pages whether they were generated from relational databases or not.

CHAPTER 3 – OVERALL APPROACH

There are three key challenges for TIBE: the user-interface, the training algorithm, and the classifier. The user-interface is important because TIBE is aimed at making it easier for an end-user to be able to create classifiers for data on web pages. The user-interface needs to be easy to learn, easy to use and intuitive to end users. It would also be nice to have the interface built so that it was plugged-in to a popular browser, rather than a program with a web browser embedded in it, so that it could be called up at any point while a user is browsing.

The main component of TIBE is the training algorithm. It is what allows a user to be able to select positive and negative training examples and train them into a classifier. The training algorithm is where much of the novel algorithms are in TIBE.

The classifier is important because how we choose to represent a classifier affects how the training algorithm may be implemented. In *Machine Learning* [Mit97] Mitchell states that by selecting a hypothesis representation, the designer of the learning algorithm implicitly defines the space of all hypotheses that the program can ever represent and therefore can ever learn. The classifier defines what hypotheses the TIBE training algorithm needs to handle and it is important that it is represented in a way that is broad enough to reliably generalize text classes, but also not so broad as to make the problem intractable.

In addition to the three main key challenges there are a few other necessary parts of TIBE. There is not necessarily anything novel about them but they are mentioned for completeness.

One feature of TIBE is that the user-interface is extensible. Other training algorithms can be plugged into the user-interface. In order to allow for extensibility, programming interfaces are defined for training and classifying text. The TIBE training algorithm is accessed through these interfaces.

The abstract programming interfaces have several requirements. They need to be abstract enough to allow for other training and classifying algorithms besides TIBE to be built on top of them. They need to be built so that the training and classification can be performed through them without knowledge of the underlying implementation.

There are two interfaces and one inheritable class that make up the programming interfaces for TIBE. The two interfaces are for a classifier domain and for a classifier. The inheritable class is a default implementation of a trainer. The classifier domain represents a collection of the training examples and classifiers for a project. A classifier for a specific text class can be requested of the classifier domain and it will return a classifier (which implements the classifier interface). The classifier interface has a method to classify text that returns a list of locations that the classifier matched. The classifier domain also has a method to get a trainer for a text class. The trainer has two methods: one to add additional training examples to the class within the classifier domain and one to get the classifier for the current text class.

In order to use this interface, an instance of the classifier domain must be created. Once the program has a classifier domain instance, it calls the method to get the trainer for a specific class. The trainer is able to accept positive and negative training examples and generate the classifier for the text class. The classifier can be passed text and return a list of text locations that it classified as the current text class. From these three programming interface elements the

user-interface or automated interface can load a project, train the classes in the project and classify text without knowledge of the underlying algorithm for training and classifying.

CHAPTER 4 – THE TIBE USER-INTERFACE

TIBE's user-interface is contained in a sidebar plug-in to Windows Internet Explorer. It has access to the web browser to be able to classify the text of the current web page and allow the user to add additional training examples. TIBE's user-interface does several things to make the training of classifiers easy for a user to do. It allows the user to straightforwardly add positive and negative training examples and gives instant classifier feedback after each training example is added so the user can plainly see what other training examples need to be added. Through the UI users can define their own lists of text classes that they want to train. It also allows the user to organize text class lists into separate projects and it auto-saves changes to the projects so that the user does not have to worry about it. Since it is a sidebar plug-in to Windows Internet Explorer, the user is able to train new or existing text classes while browsing (Figure 4-1).

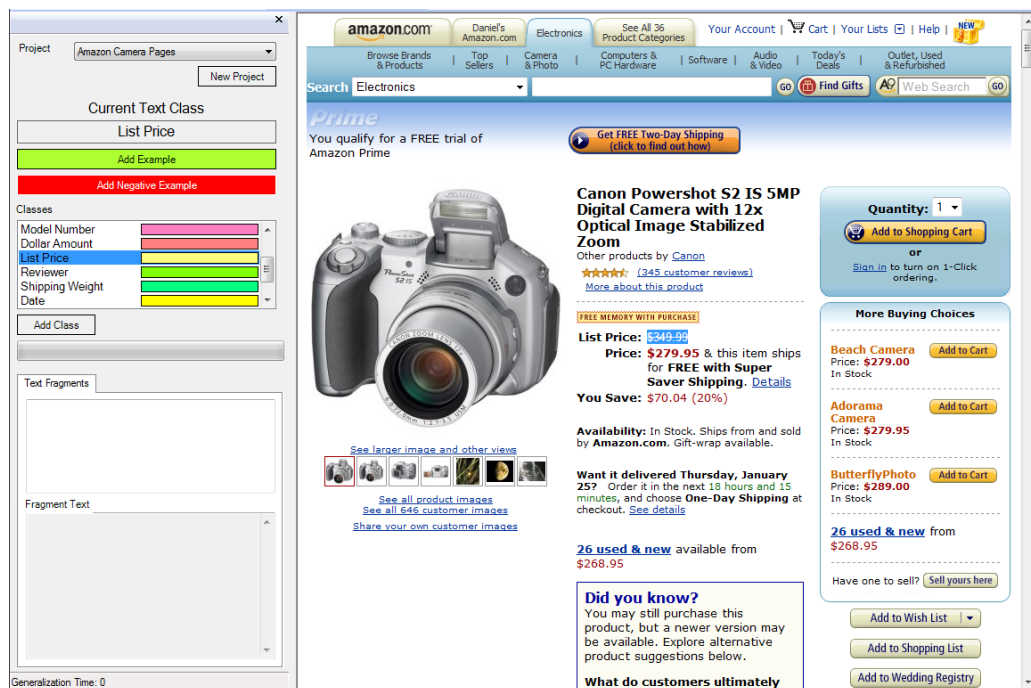


Figure 4-1: The TIBE User Interface

Implementing TIBE within a sidebar to Windows Internet Explorer has several challenges and limitations. Highlighting text that is classified changes the underlying HTML, so we make sure that training examples do not include the highlighting markup. Identifying the text and the surrounding text that was added as a training example also needs special handling, since we do not get an index to the text within the body of the HTML. We are also careful to keep the UI responsive during training, which could otherwise take several seconds at times.

The procedure to train text classes within TIBE is simple and intuitive. The user selects the class that he wants to train and starts out by selecting a positive example of that class within the current web page. The TIBE trainer will create a classifier, and the classifier will be used to highlight everything on the page that it recognizes. At this point the user will continue to add additional positive and negative examples from the page until only the positive examples are highlighted. Once this happens, he might go to another similar page on the site to see how well the classifier recognizes the class on the new page and add additional positive and negative examples. He will continue this until he has visited enough of the pages to be satisfied that the classifier is recognizing the text class correctly. The user may train another class in the class list in the same manner.

4.1 Training Example

For example, suppose a user wants to extract the “Buy New” price, “Sale Price” and the “Used & New” prices for several types of books on Amazon.com. He decides to use the book search functionality because it gives him a list of books and allows him to get the prices for multiple books on one page. The first books that he wants to search are “Harry Potter” books and so his search brings

up a listing of different “Harry Potter” books. The first thing the user does is to train dollar amounts since this is going to be the basis for the other price fields.



Figure 4-2: Training “Dollar Amount”.

In order to train the “Dollar Amount” text class, the user first selects it in the text class selection list and then selects one of the dollar amounts in the book listing web page and adds it as a positive training example (Figure 4-2). After it is added, the TIBE training algorithm generates a classifier and highlights all the other instances of “Dollar Amount” on the page that it recognizes (Figure 4-3). The user scans through the highlighted values (the “Dollar Amount” class is highlighted in green) and the rest of the page to verify that none of the “Dollar Amounts” were missed and that nothing was misclassified as a “Dollar Amount”. He also checks several other book search web pages to verify that none of them result in any misclassifications either.





- (Paperback - Jul 25, 2006)
 Buy new: ~~\$50.94~~ **\$32.09** Used & new from **\$32.09**
 Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 minutes**.
 ★★★★★
3.  **Harry Potter and the Half-Blood Prince (Book 6)** by J.K. Rowling and Mary GrandPré (Hardcover - Jul 16, 2005)
 Buy new: ~~\$29.98~~ **\$19.79** Used & new from **\$2.30**
 Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 minutes**.
 ★★★★★
 Other Editions: Hardcover, Paperback, Library Binding, Hardcover; See all 8.
4.  **Harry Potter and the Half-Blood Prince (Book 6)** by J.K. Rowling and Mary GrandPré (Paperback - Jul 25, 2006)
 Buy new: **\$9.99** Used & new from **\$4.20**
 Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 minutes**.
 ★★★★★
 Other Editions: Hardcover, Library Binding, Hardcover, Audio CD; See all 7.
5.  **Harry Potter and the Order of the Phoenix (Book 5)** by J. K. Rowling and Mary GrandPré (Paperback - Aug 10, 2004)
 Buy new: **\$9.99** Used & new from **\$3.99**
 Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 minutes**.
 ★★★★★
 Other Editions: Hardcover, Library Binding, Board book; See all 7.
6.  **Harry Potter and the Sorcerer's Stone (Book 1)** by J. K. Rowling and Mary GrandPré (Hardcover - Nov 1, 2003)
 Buy new: ~~\$24.95~~ **\$19.46** Used & new from **\$14.00**
 Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 minutes**.
 ★★★★★
 Other Editions: Hardcover, Paperback, Mass Market Paperback, School & Library Binding; See all 12.

Figure 4-3: Classified “Dollar Amounts” are highlighted in green.


2.  **Harry Potter Paperback Box Set (Books 1-6)** by J. K. Rowling (Paperback - Jul 25, 2006)
 Buy new: ~~\$50.94~~ **\$32.09** Used & new from **\$32.09**
 Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 minutes**.
 ★★★★★

Figure 4-4: Training the “Buy new” price.

Now the user is ready to train the “Buy new” price and so he selects the “Buy new” item in the text class list box and selects one of the “Buy new” prices on the book list and adds it as a positive example (Figure 4-4). This results in the TIBE trainer creating a new classifier and the classifier highlighting in yellow all the text on the page that it recognizes. This time there are several misclassifications because the training example resulted in the classifier recognizing all “Dollar Amounts” as “Buy new” prices (Figure 4-5).

2.  **Harry Potter Paperback Box Set (Books 1-6)** by J. K. F
Buy new: ~~\$50.94~~ **\$32.09** Used & new from **\$32.09**
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours** ar
★★★★★

3.  **Harry Potter and the Half-Blood Prince (Book 6)** by J.K. R
Buy new: ~~\$29.99~~ **\$19.79** Used & new from **\$2.30**
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours** ar
★★★★★
Other Editions: Hardcover, Paperback, Library Binding, Hardcover;

4.  **Harry Potter and the Half-Blood Prince (Book 6)** by J.K. R
Buy new: **\$9.99** Used & new from **\$4.20**
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours** ar
★★★★★
Other Editions: Hardcover, Library Binding, Hardcover, Audio CD; S

Figure 4-5: The “Buy new” prices that were recognized by the trained classifier.

The user recognizes that this is not right and so adds one of the misclassified amounts as a negative example. In this case, he adds one of the sale prices as a negative example (Figure 4-6).


2.  **Harry Potter Paperback Box Set (Books 1-6)** by J. K. R
Buy new: ~~\$50.94~~ **\$32.09** Used & new from **\$32.09**
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours**
★★★★★

Figure 4-6: A misclassified “Buy New” added as a negative training example.

After the TIBE trainer is done training the new classifier that resulted from the additional negative example, the “List Price” and “Used & New” prices are no longer highlighted, but there are a couple of “Buy new” prices that are not highlighted either (Figure 4-7). The user spots this and adds one of the un-highlighted “Buy new” prices as a positive example (Figure 4-8).

2.  **Harry Potter Paperback Box Set (Books 1-6)** by J. K. Rowling (Paperback)
Buy new: ~~\$50.04~~ **\$32.09** Used & new from \$32.09
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 minutes**
★★★★★

3.  **Harry Potter and the Half-Blood Prince (Book 6)** by J.K. Rowling and
Buy new: ~~\$29.99~~ **\$19.79** Used & new from \$2.30
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 minutes**
★★★★★
Other Editions: Hardcover, Paperback, Library Binding, Hardcover; See all 8.

4.  **Harry Potter and the Half-Blood Prince (Book 6)** by J.K. Rowling and
Buy new: ~~\$9.99~~ **\$9.99** Used & new from \$4.20
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 minutes**
★★★★★
Other Editions: Hardcover, Library Binding, Hardcover, Audio CD; See all 7.

5.  **Harry Potter and the Order of the Phoenix (Book 5)** by J. K. Rowling
Buy new: ~~\$9.99~~ **\$9.99** Used & new from \$3.99
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 minutes**
★★★★★
Other Editions: Hardcover, Library Binding, Board book; See all 7.

Figure 4-7: Some “Buy new” prices still have not been correctly classified.

4.  **Harry Potter and the Half-Blood Prince (Book 6)** by J.K. Rowling
Buy new: ~~\$9.99~~ **\$9.99** Used & new from \$4.20
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 minutes**
★★★★★
Other Editions: Hardcover, Library Binding, Hardcover, Audio CD; See all 7.

Figure 4-8: Another “Buy new” price is added as a positive example.

The TIBE trainer now created a classifier that appropriately classifies all of the “Buy new” prices (Figure 4-9), but it is also classifying some of the “Used & new” prices as well and so the user selects one of them and adds it as a negative example (Figure 4-10). Now the TIBE trainer creates a classifier that almost has the “Buy new” classifier working correctly.

2.  **Harry Potter Paperback Box Set (Books 1-6)** by J. K. Rowling
Buy new: ~~\$50.94~~ **\$32.09** Used & new from **\$32.09**
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 m**
★★★★★
Other Editions: Hardcover, Paperback, Library Binding, Hardcover; See all 8

3.  **Harry Potter and the Half-Blood Prince (Book 6)** by J.K. Rowling
Buy new: ~~\$29.99~~ **\$19.79** Used & new from **\$2.30**
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 m**
★★★★★
Other Editions: Hardcover, Paperback, Library Binding, Hardcover; See all 8

4.  **Harry Potter and the Half-Blood Prince (Book 6)** by J.K. Rowling
Buy new: **\$9.99** Used & new from **\$4.20**
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 m**
★★★★★
Other Editions: Hardcover, Library Binding, Hardcover, Audio CD; See all 7.

5.  **Harry Potter and the Order of the Phoenix (Book 5)** by J. K. Rowling
Buy new: **\$9.99** Used & new from **\$3.99**
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 m**
★★★★★
Other Editions: Hardcover, Library Binding, Board book; See all 7.

Figure 4-9: Current “Buy new” classifier results.


3.  **Harry Potter and the Half-Blood Prince (Book 6)** by J.K. Rowling
Buy new: ~~\$29.99~~ **\$19.79** Used & new from **\$2.30**
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 m**
★★★★★
Other Editions: Hardcover, Paperback, Library Binding, Hardcover; See all 8

Figure 4-10: Another negative example added to further train the “Buy new” classifier.

There are just two misclassified dollar amounts near the bottom of the listing. The user adds one of these misclassified fragments as a negative example (Figure 4-11) and the TIBE trainer creates another classifier. This new classifier correctly classifies the “Buy new” price on the current book listing for “Harry Potter”.

10.  **Harry Potter 2007 Wall Calendar (Calendar - Jul 1, 2006)**
Buy new: ~~\$13.99~~ **\$7.00** U
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 m**
★★★★★

11.  **Harry Potter 2007 Day-to-Day Calendar (Calendar - Jul 1, 2006)**
Buy new: ~~\$13.99~~ **\$7.00** U
Get it by **Thursday, Jan. 25**, if you order in the next **20 hours and 41 m**
★★★★★

Figure 4-11: One of the remaining misclassified amounts is added as a negative training example.

The image shows two columns of Amazon search results for books by Robert Asprin. Each item includes a book cover, title, author, and pricing information. The 'Buy new' price is consistently shown as a crossed-out value followed by a new price, which is then categorized as 'Used & new' from a specific price. This classification is correct for all items shown.

Item #	Title	Author	Buy new	Used & new
2.	Harry Potter Paperback Box Set (Books 1-6)	J. K. Rowling	\$50.04 \$32.09	from \$32.09
3.	Harry Potter and the Half-Blood Prince (Book 6)	J. K. Rowling	\$29.99 \$19.79	from \$2.30
4.	Harry Potter and the Half-Blood Prince (Book 6)	J. K. Rowling	\$9.99 \$4.20	from \$4.20
5.	Harry Potter and the Order of the Phoenix (Book 5)	J. K. Rowling	\$9.99 \$3.99	from \$3.99
6.	Harry Potter and the Sorcerer's Stone (Book 1)	J. K. Rowling	\$24.95 \$19.46	from \$14.00
5.	Concepts of Programming Languages (7th Edition)	Robert W. Bird	\$106.20 \$89.21	from \$63.00
6.	Programming Ruby: The Pragmatic Programmers' Guide, Second (Paperback - Oct 1, 2004) - Illustrated	David Flanagan	\$44.95 \$24.72	from \$17.99
7.	Programming Interviews Exposed: Secrets to Landing Your Next Job	John August	\$24.99 \$16.49	from \$13.26
8.	Programming Language Pragmatics, Second Edition	Michael D. Robbeson	\$64.95 \$45.46	from \$44.00
9.	Programming the World Wide Web (3rd Edition)	Robert W. Set	\$88.40 \$75.14	from \$73.00
2.	Class Dis-Mythed	Robert Asprin and Jody Lynn Nye	\$14.95 \$10.99	from \$10.99
3.	Another Fine Myth	Robert Asprin	\$6.99 \$2.86	from \$2.86
4.	Myth-Ing Persons / Little Myth Marker (2-In-1)	Robert Asprin	\$7.99 \$3.89	from \$3.89
1.	Security Analysis	Benjamin Graham	\$65.00 \$37.05	from \$32.39
2.	Software Security: Building Security In (Addison-Wesley)	Robert Asprin	\$49.99 \$32.99	from \$18.85
3.	Security Engineering: A Guide to Building Dependable Systems (Paperback - Jan 22, 2001)	Robert Asprin	\$75.00 \$59.25	from \$32.58
4.	Terrorism and Homeland Security	Jonathan R. White	\$61.95 \$54.52	from \$40.83
5.	Principles of Information Security	Michael E. Whitman	\$80.95 \$69.62	from \$59.99

Figure 4-12: The “Buy new” amount is being correctly classified on several additional search lists.

The user now wants to check the classifier on other search pages to verify that it appropriately classifies the text on those pages as well. In order to do this, the user searches for books by author “Robert Asprin”, books on software security and books on programming and the trained classifier correctly

recognizes the “Buy new” price on each of the resulting search pages (Figure 4-12). It was able to train the “Buy new” text class with two positive examples and three negative examples. Each of the four search pages had around 15 results with “Buy new” prices. This means that 5 training examples were able to build a classifier that was able to match around 60 instances of the text class.

Within the TIBE user-interface, a user trains one text class at a time. They specify which class they want to train by selecting it in the text class list box (Figure 4-13). The list box shows the list of current text classes available to train along with their highlight color. The selected item in the list box is the text class that is currently being trained. If a text class already has training examples associated with it and it is selected, the TIBE trainer will create a classifier from them and highlight all of the text that was recognized on the current page.

There is also a simple way to add additional classes to be trained. Defining a new class requires the user to specify the name of the class and its highlight color. A default highlight color is chosen for each new class so that the user does not need to change the highlight color unless he wants to. Once a text class is selected, the training examples already added to it are shown in another list box. This list allows the user to view what they have already trained for the current text class as well as the ability to delete training examples that may have been added in error. This allows the user to have some control over the training process and to undo mistakes within their training examples.

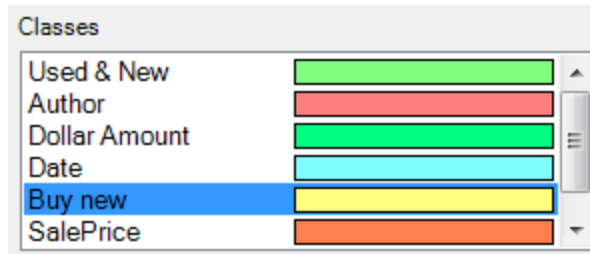


Figure 4-13: The Text class list box lists text classes that are available to train and what their highlight color is.

Since not all web pages are the same, a user is likely to want to be able to train text classes using different sets of training examples depending on the type of page. For example, training the “Buy new” price on Amazon.com is likely to require different training examples than training a similar price on barnesandnoble.com. There may also be different text classes that a user wants to train depending on the type of page. The data that a user wants to collect from a genealogy site differs from the data on a site with stock quotes or with prices on books or other electronics. In order to accommodate this, text classes can be grouped into projects within the interface. When a user opens the TIBE user-interface, he would choose the project that corresponds with the type of web page that he is working with and the text classes associated with that project would be loaded and ready to be trained.

4.2 Implementation Details

Although the TIBE user-interface has access to the HTML within the web-browser control, it is not necessarily a trivial task to highlight recognized text or to extract the training examples from user selections. In order to highlight the recognized text, an HTML element needs to be added to the markup to surround the appropriate text so that it will show up highlighted within the web browser. For example, if “\$17.37” is selected by the user as a positive text example or identified by the classifier as a “Dollar Amount”, it will be replaced with “\$17.37” in order to have the “\$17.37” show as highlighted in

the browser. Adding this additional text has the side-effect of modifying the text that is being used when adding additional training examples. Also, the classifier returns a list of indexes to the recognized text, and when the highlighting markup is inserted, the indexes of the recognized texts also change so some work has to be done to overcome this.

Extracting a training example from a user's selection is particularly tricky. Optimally we would be able to get the index and length of the selection within the body of the HTML and use that information for the training example, but this information is not provided within the selection API of the web browser control. What is available is the ability to get the HTML text of the current selection as well as the HTML surrounding the selection. Using this feature of a selection, we are able to create a fairly reliable algorithm to get the text surrounding the user's selection as well as the index and length within the surrounding text of the user's selection. Additionally we add some code to remove the TIBE highlighting markup from the surrounding text of the user's selection so that it will not be included in the training example as it would negatively bias the trained classifier.

Another feature of the user-interface is that the training of a new classifier occurs in a non-user-interface thread so that the user-interface is still responsive while it is training. We do this because sometimes the training can take a couple of seconds and we do not want the user-interface to be unresponsive during this time. We also include a visual indicator on the user-interface that the training is running so that the user knows when it is training.

CHAPTER 5 – TIBE TRAINING ENGINE

The TIBE training engine was the main challenge in making Text Identification by Example work. The TIBE training engine takes as inputs positive and negative training examples and produces a classifier that recognizes the positive examples and does not recognize the negative examples. Hopefully the classifier will also recognize other text that is part of the current text class and not misclassify text that is not the current text class. The patterns resulting from the concept learning algorithm used within the TIBE trainer are on the general boundary of the version space for the text class. We will call these patterns “minimal patterns” because they are general patterns with the minimum amount of specificity to make them match all of the positive examples, but none of the negative examples. The set of minimal patterns for a text class should not have any pattern in it that is more general than or more specific than any other pattern in the set.

We will review the rules that the TIBE training engine uses when it takes the positive and negative training examples and generalizes the positive training examples into minimal patterns. These rules determine which generalized patterns are considered minimal, which ones are generalized further and how newly generalized patterns affect the previously derived minimal patterns.

Finally, we will review several training scenarios such as training user entered dates, “List Prices”, and “Book Bindings” so that we can get a close-up look at how the algorithm works and how its rules help it make decisions at each step of the training process.

5.1 TIBE Patterns

The TIBE training algorithm depends on being able to generalize the training examples into a general classifier. The training examples are essentially very specific patterns in the version space and the general classifier's regular expression is generated from the minimal patterns for the text class. The TIBE training algorithm is what derives the minimal patterns from the positive training examples.

Since the TIBE classifier is a regular expression, the TIBE patterns could have been modeled to use regular expression syntax but this did not offer the flexibility needed to implement the TIBE training algorithm. Instead of regular expressions that are represented by a string of characters, we opted to define a set of categories that could make up a pattern. Each category can have several properties associated with it (like whether it represents multiple instances of that category) and can be easily converted into a regular expression pattern to create a classifier. In addition, each category has rules as to what is more specific or more general than it. The ability to calculate "is_more_general_than" and "is_more_specific_than" between patterns is important to the TIBE training algorithm and would have been more difficult to calculate with the regular expression syntax.

TIBE's patterns are implemented as a list of pattern elements. Each pattern element can be one of several pattern categories. The general pattern categories include literals, letters, digits and whitespace (Table 5-1). All pattern categories can be flagged as representing multiple (one or more) of the same category and Table 5-2 shows the specific-to-general relationships between the pattern categories.

Pattern Category	Symbol
Specific Literal	(Escaped Character) (ie "A" = "A" but "<" = "\<")
Any Letter	<lt>
Any Digit	<dg>
Any Whitespace	<ws>

Table 5-1: The symbolic representation of each of the pattern categories in TIBE.

In order for the TIBE training algorithm to use training examples, they must be converted into patterns. Since a training example is essentially a specific pattern of the text class, the text from a training example is converted into a pattern by representing each of the characters as a literal pattern category in its list of elements. Then as the training example (now represented as a pattern) is generalized during the algorithm, additional more general patterns are generated by replacing the literal pattern categories with other more general pattern categories as shown in Table 5-2. We support the "+" or one or more relationship, but do not support the "*" or zero or more relationship as that introduced an additional training complexity that we did not want to resolve at this time.

Specific-To-General Relationships between Pattern Categories		
Literal Letter	<lt>	<lt>+
Literal Digit	<dg>	<dg>+
Literal Whitespace	<ws>	<ws>+

Table 5-2: Pattern category specific-to-general relationships.

In addition to pattern categories listed in Table 5-1, there is a special pattern category called TIBE Text Class which is represented symbolically as the

name of the text class with “<>” surrounding it (ie <<Date>> OR <<Dollar Amount>>). The TIBE Text Class pattern category can represent any text class within the same classifier domain. A classifier domain contains all the classifiers for the text classes that the user defines as related to each other. For example, if a user was trying to train several text classes on a book page on Amazon.com, he would have a classifier domain setup in the user interface with text classes like “List Price”, “Sales Price”, “ISBN” and “Dollar Amount”. The classifiers for each of these text classes would be contained within a classifier domain. When a pattern is being generalized, other text classes are checked within the same classifier domain to see if any of them match (ie “are more general than or equal to”) elements within the current pattern. If a match is found, the matched pattern elements are replaced by a single pattern category that represents the TIBE text class within the pattern. For example, a text class “Date” which has generalized to <dg>/<dg>/<dg>+ might replace 9/12/2005, 10/14/1975, <dg>/<dg>/<dg><dg><dg><dg>, and <dg>/<dg>/<dg>+ with <<date>> within a pattern because it is “more general than or equal to” the replaced pattern elements.

Another type of generalization that can reduce the number of pattern elements within a pattern is the “multiples” generalization. When there are strings of the same generalization class, the string can be generalized to a single pattern element of the same category with its multiple flag set. For example “aaaa” could be generalized to “a+”, and “<dg><dg><dg>” could be generalized to “<dg>+”.

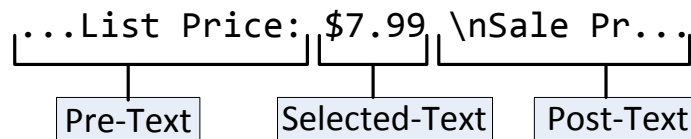


Figure 5-1: The three parts of a TIBE Training Example.

When training examples are converted to patterns for the TIBE training algorithm, the text preceding and following the selected text is also included in the pattern. A TIBE pattern consists of three parts: the text preceding the training example (pre-text), the text which is the training example (selected-text), and the text following the training example (post-text) (Figure 5-1). The fragment boundary pattern categories (represented symbolically as “<frag>”) are used to separate the pre-text, selected-text and post-text parts of the pattern so that when the patterns is compared to other patterns or converted to classifiers, the different parts can be handled appropriately (Figure 5-2).

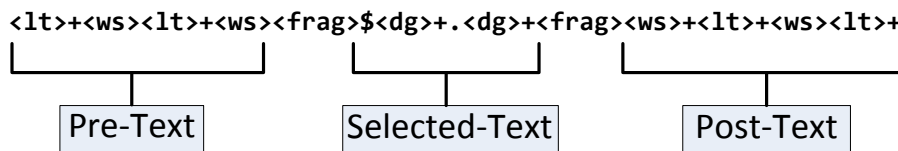


Figure 5-2: The Pre-Text, Selected-Text and Post-Text of a TIBE pattern.

The pattern “is more general than” test compares two patterns by first splitting the patterns into their three parts and testing each part separately. If any part fails the “is more general than or equal to” test, the whole training example fails the test.

There are some specific rules that apply to the parts of a pattern that represent the pre-text or post-text of a training example. For these, a pattern part “is more general than or equal to” another pattern part if it is shorter than the longer pattern part and the shorter pattern part is “more general than or equal to” the longer pattern part for the length of the pattern part. For pre-text the shortness applies to the left side of the pattern part and for the post-text it applies to the right side of the pattern part. Table 5-3 shows examples of Pre-Text and Post-Text “is more general than” comparisons. In each example, Pattern 1 “is more general than” Pattern 2.

Pre-Text		Post-Text	
Pattern 1	Pattern 2	Pattern 1	Pattern 2
Date:<frag>	Birth Date:<frag>	<frag>year	<frag>years old
<lt>:<frag>	Birth Date:<frag>	<frag><lt>	<frag>years old
<ws><lt>+:<frag>	Birth Date:<frag>	<frag><lt>+<ws>	<frag>years old
th:<frag>	Birth:<frag>	<frag>	<frag>years old
th:<frag>	Death:<frag>	<frag>\</<lt+>	<frag>\</font\>

Table 5-3: A more-general-than comparison between pre-text and post-text pattern parts.

Another important comparison between two patterns is the “is more specific than” comparison. The “is more specific than” comparison is performed by reversing the “is more general than” comparison on the two patterns. For example, if P_1 and P_2 are two patterns and P_1 “is more general than” P_2 , then conversely, P_2 “is more specific than” P_1 .

5.2 Converting a Pattern into a Classifier

Once a training example has been trained into a minimal pattern, it needs to be converted into a regular expression so that it can become a classifier for the current class. Converting a pattern into a regular expression is fairly straightforward. With the exception of a single special-case, the conversion of a pattern category is direct (See Table 5-4). The single exception is for the TIBE text class pattern category which gets converted into the regular expressions that make up its classifier. For example, if the <<Date>> text class is embedded in the death date text class (for example: ath: <<Date>>), the <<Date>> text class would be converted to the full regular expression that makes it up rather than just a single regular expression like the other pattern categories. There are also mechanisms to prevent recursive TIBE text classes within a generalization to prevent endless loops when expanding TIBE text class pattern categories into their classifiers.

Pattern Category	Regular Expression
Literal (ie abc, 123, \[])	Literal (Possibly escaped) (ie abc, 123, \\[\])
Letter	[A-Za-z]
Digit	\d
Whitespace	\s
Multiple	[A-Za-z]+, \d+, \s+

Table 5-4: The conversion from generalization classes into regular expressions is straightforward.

Since a pattern has a pre-text, selected-text and a post-text part, each part is converted to its own regular expression separately and then combined into one regular expression. The pre-text regular expression is put into a zero-width positive look-behind assertion group (represented as `(?<=)`) and the post-text regular expression is put into a zero-width positive look-ahead assertion (represented as `(?=)`). These regular expression constructs require that the regular expression within their groups be matched on the left or right of the current position without including those characters in the match. Figure 5-3 demonstrates how a minimal pattern is converted into a regular expression.

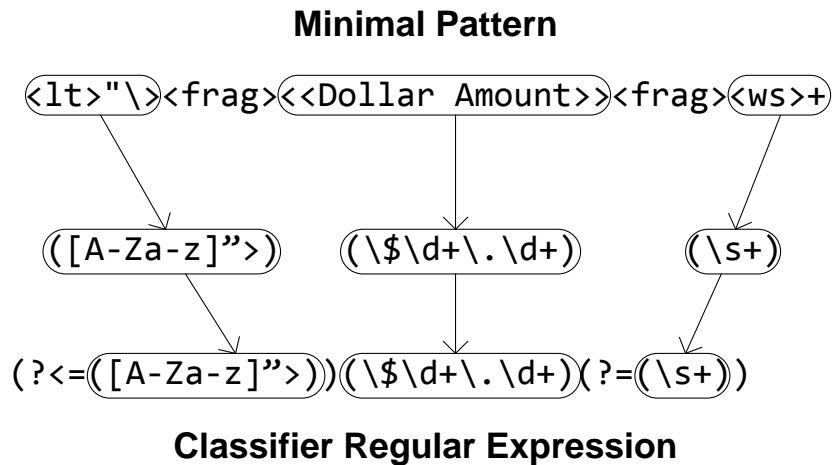


Figure 5-3: Conversion from a minimal pattern into a classifier regular expression.

5.3 TIBE Training Algorithm

The TIBE training algorithm takes the list of positive and negative training examples and creates two separate lists for them. The positive training examples need to be generalized and the positive and negative training examples need to be referenced by the generalization algorithm. One at a time, each positive training example is sent through the generalization algorithm to get their list of minimal patterns. When the generalization is completed on a positive training example, its set of minimal patterns is compared against each of the positive training examples that remain and compared against the minimal patterns that have already been generated.

When comparing against the other positive training examples, if the minimal patterns are more general than a positive training example, the training example is removed from the list of training examples to generalize because its minimal patterns would be “more specific or equal to” the current minimal patterns and so there is no need to generalize it. For example suppose the positive training examples were three dates like “12/30/2005”, “4/4/1977” and “1/29/1985”. If “12/30/2005” was

the first training example to be generalized, it would generalize to “<dg>+<dg>+<dg>+” which is more general than the other two training examples. The other two training examples would generalize to “<dg>/<dg>/<dg>+” and “<dg>/<dg>+<dg>+” respectively and both of those minimal patterns are more specific than “<dg>+<dg>+<dg>+” and so we see that we would not need to generalize them. This is one way that the algorithm attempts to reduce the amount of work necessary in order to generate minimal patterns for a set of training examples.

The minimal patterns are also compared against the other minimal patterns that have been generated previously. Any of the previous minimal patterns that are more specific than the current patterns are removed since they would no longer be considered minimal for the text class. For example if the positive training examples were three dates like “12/30/2005”, “4/4/1977” and “1/29/1985” but this time “12/30/2005” was generalized last instead of first, then when it was generalized to “<dg>+<dg>+<dg>+”, the minimal pattern would be compared to “<dg>/<dg>+<dg>+” because it would be the current minimal pattern. The minimal pattern “<dg>/<dg>+<dg>+” would be replaced by “<dg>+<dg>+<dg>+” because it is more specific than the new minimal pattern. If another positive training example was added as “Aug. 21, 1999”, it would generalize to “<lt>+. <dg>+, <dg>+” which is neither more general nor more specific than “<dg>+<dg>+<dg>+” and so would be added as another minimal pattern in the text class’s set of minimal patterns. Although this part of the algorithm does not necessarily reduce the amount of work performed to calculate the minimal patterns, it ensures that we do indeed end up with a list of minimal patterns and it makes it so that the order in which the training examples are generalized does not matter in calculating the minimal patterns.

5.4 Pattern Generalization

Within a concept learning algorithm like the TIBE training algorithm, there is a hypothesis space that consists of all patterns that could represent a text class. For a TIBE pattern, this would be all the combinations of literal, letter, digit, whitespace and TIBE text class pattern generalizations. It also includes combinations of these with their multiple options set and differing numbers of them within the pre-text, selected-text and post-text parts of the pattern. The number of these combinations can easily be a very large number, but when generalizing a training example, we only need to work with the small subset of these which are in the version space of the training example. Figure 5-4 shows what the version space might look like for a date like “10/31/1999”. The version space is the patterns that exist between the specific boundary and the general boundary. In this simple example the specific boundary is “10/31/1999” and the general boundary is “<dg>+</dg>+</dg>+”.

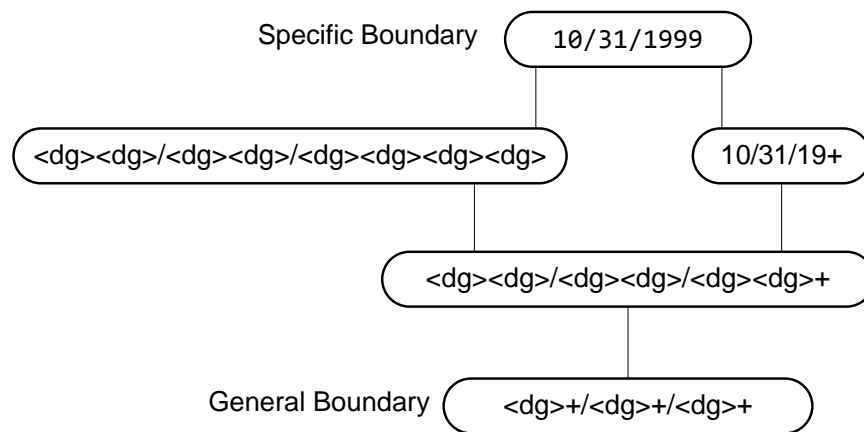


Figure 5-4: The patterns that exist within the version space for “10/31/1999”.

Adding negative training examples generally makes the general boundary more specific. For example if we were training “Birth Date” we might add “Birth: 4/4/1977” as a positive training example. This would generalize to “<<Date>>” (assuming that the “Date” text class has already been trained) which would

match all dates instead of just birth dates. A negative training example of "Marriage: 8/21/1999" could be added as a negative example which would cause the birth date text class to generalize to "h:<ws><<date>>" which no longer match dates that do not start with "h:<ws>" but there are still some dates that are not birth dates that start with "h:<ws>" like "Death: 9/10/2006". When "Death: 9/10/2006" is added a negative example, the birth date generalizes to "rth:<ws><<date>>" which now only matches dates that start with "rth:<ws>" which is still general enough to find several birth dates, but certainly more specific than the original minimal pattern of "<<date>>".

Adding negative training examples can also result in multiple minimal patterns on the general boundary. For example suppose we were training birth dates like in the previous example. In those examples we only had pre-text but usually there is also post-text in the example as well. The first positive training example might be "Birth: 4/4/1977\r" which would generalize to "<<date>>" just like it did originally. The first negative training example might be something like "Marriage: 8/21/1999 " which would cause the text class to generalize to "h:<ws><<date>>" and "<<date>>\r". Both patterns will match the birth date as well as other dates that start with "h:<ws>" or end with "\r". In this case there might be dates like "Death: 9/10/2006 " that are matched because they start with "h:<ws>" or dates like "Baptism: 5/4/1985\r" that are matched because they end with "\r". Both of these would need to be added as negative training examples since they are not birth dates. The resulting minimal pattern would now be "rth:<ws><<date>>" just like before.

Adding positive training examples can reduce the number of general boundary patterns if they are similar to other positive training examples. In the previous example, after a positive and a negative training example were added, the resulting minimal patterns were "h:<ws><<date>>" and "<<date>>\r". If another positive training example of "Birth: 7/2/1980 " was added, the minimal patterns

would be reduced to just “h:<ws><<date>>” because it matches more of the positive examples than “<<date>>\n”. Thus, it is given more weight in the generalization algorithm.

If a positive training example is different from the other positive training examples, it can create a separate pattern space that must be generalized. This would occur in the date example if another positive training example was “Apr. 23, 2005”. Figure 5-5 shows the version space for this type of date. Because its version space does not overlap the “mm/dd/yyyy” version space, it just adds the additional minimal pattern “<lt>+.<ws><dg>+,<ws><dg>+” to the set of minimal patterns for the “Date” text class.

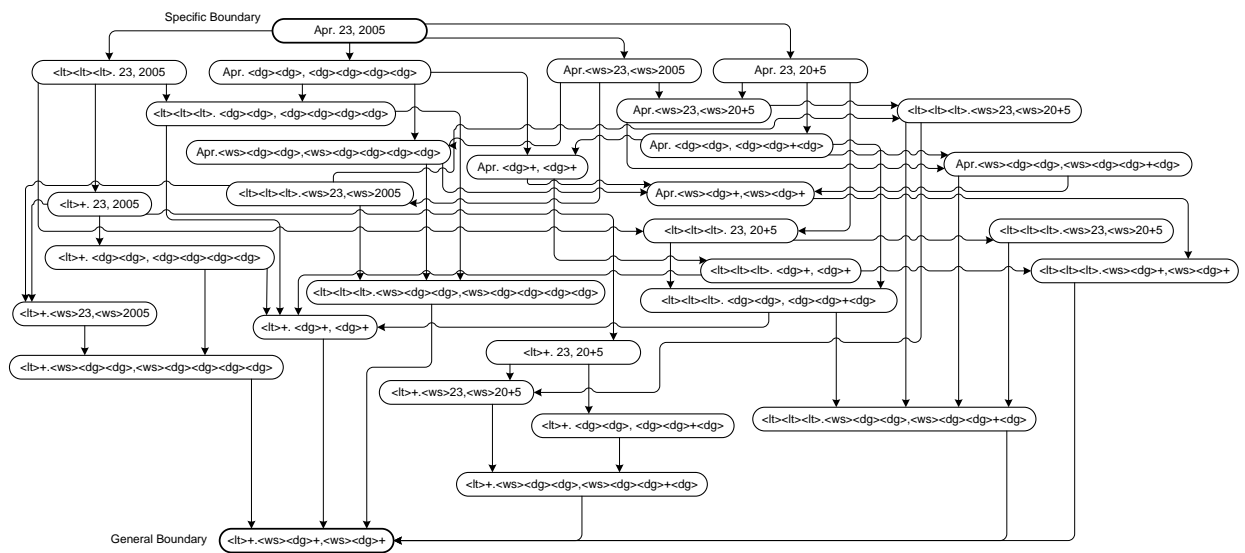


Figure 5-5: Version space for “Apr. 23, 2005”.

In order to obtain the general boundary of a text class’s version space, usually we start with the most general pattern in the pattern space and then make it more specific until it still matches the positive training examples, but not the negative training examples. Technically, the most general pattern would be one that matches anything, but the TIBE pattern space does not have a way to

represent a “match anything” pattern and so the most general pattern for a training example is the pattern that results from applying the TIBE generalization rules on a pattern until no more general patterns are generated from a pattern. In the date example, “10/31/1999” is not the general pattern because when the generalization rules are applied to it, two more general patterns result: “10/31/19+” from the “strings” generalization and “<dg><dg>/<dg><dg>/<dg><dg><dg><dg>” from the digit generalization. Neither of these patterns are the most general pattern because further generalization can occur on both. Eventually generalization of these patterns will result in a pattern like “<dg>+/<dg>+/<dg>+”, which cannot be generalized further using the TIBE generalization rules. This is the most general pattern for the training example “10/31/1999”.

5.4.1 Deriving the minimal pattern

The most general pattern is important to the concept learning algorithms like TIBE because they start with the most general pattern and then test it against the negative training examples and make the most general pattern more specific if it matches any of them. In the TIBE algorithm, we do not implicitly know the most general pattern of a given training example and so must derive it by generalizing the training example. While deriving the most general pattern, if a pattern is found to match a negative example, generalization on that pattern is stopped and its parent pattern is added to a list of possible minimal patterns. For example, if we were training a year with a positive training example of “1985”, it would generalize to “<dg><dg><dg><dg>” and then to “<dg>+”. If we had a negative training example of “Pg. 35”, this would cause the generalization to stop at “<dg><dg><dg><dg>” because “<dg>+” matches the “35” in “Pg. 35” and so “<dg><dg><dg><dg>” would be added as a minimal pattern for the “Year” text class. If there were not any negative training examples, or the most general pattern did not match any of them, then the most general pattern becomes one of the minimal patterns for the text class.

The minimal pattern for a positive training example is found by trying to derive its most general pattern. The pre-text and post-text are initially removed from the positive training example's initial pattern since the selected-text alone is more general than the selected-text with its pre-text and post-text. For instance, suppose a positive training example for "List Price" is "List Price: \$15.98". The selected-text "\$15.98" is more general than the full training example and the TIBE algorithm starts by generalizing it. If there were no negative training examples, this would result in the most general pattern being generated as: "\$<dg>+.<dg>+". However, if there was a negative training example of "Sales Price: \$10.98", then the generalization of the selected text would not have proceeded past "\$15.98" because its first generalization "\$<dg><dg>.<dg><dg>" would have matched the "Sales Price" dollar amount. In addition, since it matched a negative training example, it is not generalized further because all subsequent generalizations would also match the negative training example. So the minimal pattern generated so far is "\$15.98" which is more general than the positive training example with its pre-text and post-text, but it is not likely to be a very good classifier.

5.4.2 Making the general boundary more specific

When generalizing just the selected-text portion of a positive training example, the TIBE algorithm is considered to be in level 0 of the generalization. If any negative training examples were matched during a level's generalization, the algorithm will proceed on to the next level of generalization. The next level generalizes more specific versions of the positive training example by adding additional elements from its pre-text and post-text. Table 5-5 shows the levels 0 through 4 seed patterns for the positive training example "List Price: \$15.98\nSales".

Level 0	Level 1	Level 2	Level 3	Level 4
\$15.98	\$15.98	: \$15.98	e: \$15.98	ce: \$15.98
	\$15.98\n	\$15.98\n	: \$15.98\n	e: \$15.98\n
		\$15.98\nS	\$15.98\nS	: \$15.98\nS
			\$15.98\nSa	\$15.98\nSa
				\$15.98\nSa1

Table 5-5: The level 0 through level 4 pattern seeds for “List Price: \$15.98\nSales”.

The TIBE algorithm stops generalizing a training example once it reaches a level where none of its generalizations matched a negative example. Since none of the generalizations at that level matched a negative training example, we know that any more specific patterns would also not match a negative training example and since we are trying to derive the general boundary in the pattern space, we know that we do not need to generalize any more specific versions of the positive training example.

5.4.3 TIBE Training Rules

So far we have discussed how the most general boundary of a level in the pattern generalization is calculated and how we determine the highest level to generalize, but in this process there are several rules that determine which of the intermediate generalized patterns are added to the list of minimal generalizations for a positive text example. At each step of generalization, the generalized pattern is checked against the list of generalized patterns that have already been generated. As can be seen in the version space for “Apr. 23, 2005” in Figure 5-5, several intermediate patterns may generalize to the same generalized pattern, but we only need to run the generalization on that new generalized pattern once. If the generalized pattern has not been generated previously, then it is added to a stack of patterns to generalize further.

After generating all generalized patterns from a pattern, the top of the “To-Generalize” pattern stack is popped off. Before we generalize this next pattern,

we first check it against the list of already generalized patterns in case a similar pattern had been generalized after this one had been added to the “To-Generalize” pattern stack. If it has not been generalized yet, then we add it to the list of already generalized patterns in anticipation of its generalization. Next, the TIBE generalization algorithm attempts to add the current pattern to the list of minimal patterns for the current positive training example. The result of this determines whether the pattern will be further generalized or not.

Adding a pattern to the list of minimal patterns for the current positive training examples consists of several checks that determine whether the pattern should be added and whether the pattern should be generalized further. The first check is to see if the pattern matches any of the negative training examples. If it does, then it indicates that this pattern is too general and any further generalizations of it would also match the negative example and so we do not add it as a minimal pattern and we indicate that it should not be generalized further. For example, suppose that a “Sales Price” was being trained and within the algorithm, the current pattern was “\$<dg><dg>.<dg><dg>” and that one of the negative training examples was a “List Price” that looks like “List Price: \$14.95”. The current pattern matches the negative training example and so it is not added as a minimal pattern and is not generalized further because any more general pattern would still match the negative example. The next generalization of the pattern would have resulted in “\$<dg>+.<dg>+” which still matches the negative example.

If the pattern did not match any of the negative training examples, it becomes a minimal pattern candidate but we still have a few more checks to make. First we compare the pattern against the positive training examples giving more weight to patterns that match more positive examples. If a pattern matches more positive examples than another pattern that is currently in the list

of minimal patterns for the current positive training example, then it replaces that pattern in the list and the pattern is generalized further. For instance, a positive training example for a birth date might be “..was born on 4/4/1977 in Provo, UT..”, which given the right negative training examples might have as its minimal patterns: “born on <<date>>” and “<<date>> in”. If another positive training example of “..born on 7/2/1980. The hospital..” was added to the text class, then this rule would cause the “born on <<date>>” to replace “<<date>> in” as a minimal pattern because it matches more positive training examples and so is more likely to create a classifier that will match more of the text class.

If the pattern matches fewer positive training examples than the current minimal patterns, then it is not added to the list of minimal patterns, but it is still generalized further. For example, similar to the previous example, suppose the positive training examples for birth date are “..was born on 4/4/1977 in Provo, UT..”, “..born on 12/30/2005 on a cold night..” and “..born on 7/2/1980. The hospital..”. Assuming the appropriate negative training examples were added to make it happen, the current pattern being evaluated might be “born on <dg><dg>/<dg><dg>/<dg><dg><dg><dg>” and the current minimal pattern might be “<<date>> <lt><lt>”. The current minimal pattern matches two of the positive training examples and the pattern being evaluated only matches one of them. But if the pattern is further generalized, it could result in a pattern “born on <<date>>” which matches all three positive training examples and would replace “<<date>> <lt><lt>” in the list of minimal patterns for the positive training example.

If the pattern matches the same amount of positive training examples as the current minimal patterns, then it replaces any of the current minimal patterns that it is more general than and is generalized further. This rule makes sure that “<dg><dg>/<dg><dg>/<dg><dg><dg><dg>” replaces “12/30/2005”, “<dg>+<dg>+<dg>+” replaces

“<dg><dg>/<dg><dg>/<dg><dg><dg><dg>” and that “<<date>>” replaces “<dg>+<dg>+<dg>+” as minimal patterns. Since we are trying to derive the general boundary of the version space, if there is a more general pattern than what is currently in the minimal pattern list, then it will replace the more specific versions and we want to continue to generalize it further to see if a more general pattern can be found.

If it is not more general than any of the current minimal patterns, it will be added to the list of minimal patterns and generalized further if none of the minimal patterns are more general than it. However, if any of the minimal patterns are more general than it, then it does not get added to the list minimal patterns for the current positive training example since a more general pattern has already been derived that does not match a negative training example. When not many positive training examples have been added to aid in training, several alternate minimal patterns often result. For example, if “List Price” is being trained and the positive training example is “List Price: \$12.99\n” and a negative training example might be “..the item is sold for \$11.25 at the auction..”. The trainer is going to first try to generalize to “<<dollaramount>>” but that matches the negative training example and so the algorithm proceeds from level 0 into level 1. At this level both “:<ws><<dollaramount>>” and “<<dollaramount>>\n” are minimal patterns since both match the positive training example and neither are more general than the other. Now let’s assume that another negative training example of “Discount: \$3.00 off ” is added. This would cause the “:<ws><<dollaramount>>” pattern to match a negative training example and need to be made more specific. The result would be “e:<ws><<dollaramount>>” as one of the minimal patterns. The other pattern “<<dollaramount>>\n” would remain as a minimal pattern because it would be more general than any of the more specific patterns that are tested against it such as “<ws><<dollaramount>>\n”, “:<ws><<dollaramount>>\n” OR “e:<ws><<dollaramount>>\n”.

When a pattern is found to be more specific than one of the current minimal patterns, it will be generalized further if it is composed of all literals. This due to the fact that while testing, there were several cases where generalization stopped prematurely if the literal pattern was not generalized further. For example, suppose that “list price” was being trained again and “List Price: \$12.99\n” and “List Price: \$8.59\n” are positive training examples and “Sales Price: \$9.99\n” is a negative training example. Within the level 0 generalization, the first positive training example would generalize to “\$<dg><dg>.<dg><dg>” because both “\$<dg>+.<dg>+” and “<<dollaramount>>” would match a negative training example. Generalization would stop immediately on the next level without this additional rule. This is because “\$<dg><dg>.<dg><dg>” is more general than all other combinations of “\$12.99” and its pre-text and post-text, therefore generalization would stop with the first seeds of “ \$12.99” and “\$12.99\n”, not giving it a chance to match a negative training example and continuing on. With the addition of this rule, generalization would continue on each level, because although “ \$12.99” would not be added as a minimal pattern, it would be further generalized to “ <<date>>” which would match a negative training example and force the algorithm to continue on to level 2. Each level would match a negative training example until it came across the pattern “t Price: <<dollaramount>>” which does not match a negative training example and actually matches both positive training examples and so would remove “\$<dg><dg>.<dg><dg>” from the list of minimal patterns because it only matched one positive training example. We suspect that in addition to generalizing further if the pattern is all literals, we might always want to generalize further when a pattern is more specific than a current minimal pattern but could not test the impact that this would have on the algorithm for this thesis.

5.5 TIBE Trainer Scenarios

In this section we will review several different training scenarios and what the TIBE training algorithm would do in each one. The first scenario is training dates on a genealogy site where the entries are human entered and so the dates come in several different formats. The second scenario is training a “List Price” on a set of pages on Amazon.com related to digital cameras. In each example, we are going to assume that all the necessary positive and negative training examples have been already added in order to train the correct classifier. In this section we are more interested in how the algorithm derives the appropriate classifiers given the correct positive and negative examples, rather than the process by which those classifiers are chosen by the user. We will also assume that other text classes that might be used by the current text class have already been trained. For example “<<dollaramount>>” will already have been trained when training the “List Price” text class. Although each of these scenarios are based upon actual training scenarios on real web pages, we are going to simplify the training examples in order to explain how the algorithm works without being weighed down by all the gory details.

5.6 Training Dates on a Genealogy site.

This example demonstrates the training of dates on a genealogy site where people enter genealogical queries. Most entries on the site are by people looking for more information on a relative and often have dates related to birth dates and death dates. Because the entries are entered by humans, the dates are entered in a variety of different formats. This example will show how the TIBE training algorithm handles a variety of different positive training examples for the same text class. The dates trained in this example also do not need any pre-text or post-text in order to make the patterns specific enough to not match any negative training examples.

The demonstration of the training of this text class is split into three sections. The first section just takes one of the positive training examples and demonstrates how it was generalized. The second section demonstrates what happens when a generalized pattern matches more than one positive training example. The final section will demonstrate what happens when the generalization of the training example matches a negative training example.

5.6.1 Generalizing a "Date" Positive Training Example

In the process of generalizing the "Date" text class, several positive and negative training examples have been added to fully train the classifier. In generalizing the "Date" text class, the outer loop of the training algorithm begins by attempting to generalize each positive training example in order to get their minimal patterns. Generalization of "Posted: <frag>09-Sep-2006</frag>" begins by generalizing just the selected-text portion of the example: "09-Sep-2006". The first possible minimal pattern that is added for this positive training example is "09-Sep-2006" because there are no other possible minimal patterns that are more general than it and it does not match any negative training examples.

After "09-Sep-2006" is added to the list of possible minimal patterns, it is generalized by applying the letter, digit, whitespace and multiple character generalizations to it. The multiple character generalization is applied first and results in a new more general pattern of "09-Sep-20+6" which is placed on the "To-Generalize" stack of patterns to be tested for inclusion in the list of minimal patterns. The whitespace generalization is applied next, but because there is not any whitespace in this pattern, the generalization results in the same pattern. Next the digit generalization is applied resulting in a more general pattern of "<dg><dg>-Sep-<dg><dg><dg><dg>" which is placed on the "To-Generalize" stack. The last

generalization performed on the initial pattern is the letter generalization resulting in “09- $\langle \text{lt} \rangle \langle \text{lt} \rangle \langle \text{lt} \rangle$ -2006” which is also placed on the “To-Generalize” stack.

The “To-Generalize” stack now has 3 patterns on it that need to be generalized further and the current possible minimal pattern is “09-Sep-2006”. Since generalization on the previous pattern is complete, generalization proceeds on the next pattern that is popped from the top of the “To-Generalize” stack which results in the pattern “09- $\langle \text{lt} \rangle \langle \text{lt} \rangle \langle \text{lt} \rangle$ -2006”. This pattern is first checked to see if it is a possible minimal pattern. During this check it is checked against the negative training examples and does not match any of them. Then it is checked against the positive training examples to give it more weight if it matches more positive training examples. In this case it only matches one positive training example (the positive training example that it was generalized from). Next it is compared against each of the current possible minimal patterns to see if it is more specific or more general than any of them. It is more general than the current possible minimal training example of “09-Sep-2006” and so replaces it in the list of minimal training examples.

Since “09- $\langle \text{lt} \rangle \langle \text{lt} \rangle \langle \text{lt} \rangle$ -2006” was added as a possible minimal training example, it is generalized further. Applying the standard generalizations to this pattern result in two additional patterns being added to the “To-Generalize” stack: “09- $\langle \text{lt} \rangle$ -20+6” and “ $\langle \text{dg} \rangle \langle \text{dg} \rangle$ - $\langle \text{lt} \rangle \langle \text{lt} \rangle \langle \text{lt} \rangle$ - $\langle \text{dg} \rangle \langle \text{dg} \rangle \langle \text{dg} \rangle$ ”.

Generalization continues with “ $\langle \text{dg} \rangle \langle \text{dg} \rangle$ - $\langle \text{lt} \rangle \langle \text{lt} \rangle \langle \text{lt} \rangle$ - $\langle \text{dg} \rangle \langle \text{dg} \rangle \langle \text{dg} \rangle$ ” being popped from the “To-Generalize” stack and checked as a possible minimal pattern. It does not match any negative training examples and matches only one positive training example and is more general than the only other pattern in the possible minimal patterns list and so replaces “09- $\langle \text{lt} \rangle \langle \text{lt} \rangle \langle \text{lt} \rangle$ -2006” as a possible minimal pattern. When this new pattern is generalized only one more pattern is added to

the “To-Generalize” stack: “<dg>+<lt>+<dg>+”. When “<dg>+<lt>+<dg>+” is popped from the “To-Generalize” stack and checked as a possible minimal pattern, it replaces “<dg><dg>-<lt><lt><lt>-<dg><dg><dg><dg>” as a possible minimal pattern because it is more general.

Generalizing “<dg>+<lt>+<dg>+” results in no more general patterns to add to the “To-Generalize” stack and so the next pattern popped from the stack is “09-<lt>+20+6”. When this pattern is checked as a possible minimal pattern, it is found to be more specific than the current possible minimal pattern “<dg>+<lt>+<dg>+” and so is not added or generalized further. Each of the remaining items on the “To-Generalize” stack (“<dg><dg>-Sep-<dg><dg><dg><dg>” and “09-Sep-20+6”) are also found to be more specific than the current possible minimal pattern and so are not added or generalized further either. Ultimately, the “To-Generalize” stack becomes empty indicating that all generalizations have been performed for this level of generalization. Since no negative training examples were matched during the generalization of this level, generalization does not need to continue to level 1 and so the current possible minimal patterns can be returned as the minimal patterns for the current positive training example. In this case, “<dg>+<lt>+<dg>+” is what is returned as the minimal pattern for the positive training example “Posted: <frag>09-Sep-2006<frag>”.

Within the outer loop of the TIBE trainer, before the next positive training example is generalized, the minimal pattern “<dg>+<lt>+<dg>+” is added to the list of minimal patterns for the text class and then compared against the positive training examples that have not been generalized yet. It matches the positive training example that it was derived from and so that positive training example is removed from the list of training examples that still need to be generalized and then the next item in that list is put through generalization.

5.6.2 Matching more than one positive training example

In the list of positive training examples for the “Date” text class, sometimes there would be two dates with similar structure. For example, suppose “s born on <frag>March 5, 1972<frag> at Cott” was trained first in the list of positive training examples. Its minimal pattern when generalized would be “<lt>+<ws><dg>, <ws><dg>+”. Subsequently, the positive training example “n on <frag>November 28, 1968<frag>. It” is generalized. This positive training example is similar in structure to the previous one, but has a two digit day instead of just a single digit. This positive training example eventually generalizes to the minimal pattern “<lt>+<ws><dg>+, <ws><dg>+” which matches these two positive training examples. When this minimal pattern is returned to the outer loop, it still only removes its positive training example from the list of patterns to generalize since the other positive training example that it matches (“s born on <frag>March 5, 1972<frag> at Cott”) has already been generalized and removed previously. However, when its minimal pattern is compared against the current list of minimal patterns for the current text class, it is found to be more general than “<lt>+<ws><dg>, <ws><dg>+” and so replaces it in the list.

5.6.3 Matching a Negative Training Example

Some of the positive training examples for “Date” result in more than one minimal pattern. Suppose the positive training example is “n <frag>23 April 2006<frag>. s” and the negative training examples are “n <frag>1972 or 1973<frag>. ” and “19<frag>55 or 1956<frag>. ”. After applying several generalizations to this positive training example, eventually the pattern “<dg><dg><ws><lt><lt><lt><lt><lt><ws><dg><dg><dg><dg>” is encountered. The only generalization that can still be applied to this pattern is the “string” generalization which results in the pattern “<dg>+<ws><lt>+<ws><dg>+” which matches the negative training example “n <frag>1972 or 1973<frag>. ”. This prevents this pattern from being a minimal pattern so the prior pattern remains in the list of minimal patterns.

Within the pattern space of this positive training example, the “23<ws><lt>+<ws>20+6” pattern occurs and is neither more general nor more specific than the current minimal pattern of “<dg><dg><ws><lt><lt><lt><lt><lt><ws><dg><dg><dg><dg>”. If this pattern is generalized further by applying the “digit” generalization, the pattern “<dg><dg><ws><lt>+<ws><dg><dg>+<dg>” results which matches the negative training example “19<frag>55 or 1956<frag>.”. Since a negative training example is matched, this pattern cannot be considered minimal and so the previous pattern is added to the list of minimal patterns for this positive training example resulting in two minimal patterns: “<dg><dg><ws><lt><lt><lt><lt><lt><ws><dg><dg><dg><dg>” and “23<ws><lt>+<ws>20+6”. The list of resulting minimal patterns that were trained is listed in Table 5-6.

Minimal Patterns for the “Date” Text Class
<dg>+<lt>+<dg>+
<lt>+<ws><dg>+, <ws><dg>+
<dg><dg><ws><lt><lt><lt><lt><lt><ws><dg><dg><dg><dg>
23<ws><lt>+<ws>20+6

Table 5-6: The minimal patterns for the “Date” text class.

5.7 Training List Price on Camera Pages

Training the list price on an Amazon.com camera page demonstrates a few more of the features of the TIBE training algorithm. In the previous example with the dates, we were training an element that naturally generalizes to something that is general enough to match text that is similar to it, but specific enough to not match text that is not part of the text class of date. Because of this, the TIBE algorithm did not need to include any pre-text or post-text in its minimal patterns. This is more likely to happen when there are a variety of numbers, letters and symbols within the selected-text of the text-class. In this example we still have a selected-text that naturally generalizes (“Dollar Amount”), but we are trying to train a more specific variation called “List Price”.

This means that the classifier is going to need pre-text and/or post-text in order to specify what “Dollar Amounts” are of the “List Price” class.

The HTML within web pages adds additional markup that often helps in the classifying of text classes. Although HTML is useful in this respect, it is not necessary in order to demonstrate the algorithm, and so the positive and negative training examples that we list for our demonstration will not include any HTML. The set of positive and negative training examples for this demonstration are listed in Table 5-7. The negative training examples represent several of the cases where there was a dollar amount that was not a list price.

Positive Training Examples		Negative Training Examples	
1	List Price: <frag>\$199.99<frag> \n\n	1	Sale Price: <frag>\$25.99<frag> \n\n
2	List Price: <frag>\$37.25<frag> \n\n	2	on Price: <frag>\$83.99<frag>\n
		3	s \$175 to <frag>\$199.99<frag>.\n

Table 5-7: The Positive and Negative Training Examples for the “List Price” text class.

The outer loop of the algorithm starts generalizing the first positive training example. The level 0 generalization of “List Price: <frag>\$199.99<frag> \n\n” starts with just the selected-text “\$199.99” which immediately matches negative training example #3 and so generalization at that level halts and the algorithm continues on to the level 1 generalization.

During the level 1 generalization the pattern “ <frag>\$199.99<frag>” immediately matches negative training example #3 and so is not generalized any further but “<frag>\$199.99<frag> ” is added as a possible minimal pattern. It is soon replaced by the more general “<frag>\$199.99<frag><ws>”. This can be further generalized to “<frag><dg><dg><dg>.<dg><dg><frag><ws>” without matching a negative training example, but when this is generalized to “<frag><dg>+.<dg>+<frag><ws>” OR “<frag><<Dollar Amount>><frag><ws>”, the patterns match negative training examples #1 and #2.

Since negative training examples were matched during the level 1 generalization, the algorithm continues on through the level 2 generalization. At this point, “<frag>\$<dg><dg><dg>.<dg><dg><frag><ws>” is the only pattern currently in the list of minimal patterns for this text class. The next pattern to be added to the list of minimal patterns is “:<ws><frag>\$<dg><dg><dg>.<dg><dg><frag>”. Further generalization of this pattern results in “:<ws><frag>\$<dg>+.<dg>+<frag>” and “:<ws><frag><DollarAmount><frag>” which both match negative training examples #1 and #2.

Level 3 Seeds		Current Minimal Patterns	
1	e: <frag>\$199.99<frag>	1	<frag>\$<dg><dg><dg>.<dg><dg><frag><ws>
2	: <frag>\$199.99<frag>	2	:<ws><frag>\$<dg><dg><dg>.<dg><dg><frag>
3	<frag>\$199.99<frag> \n		
4	<frag>\$199.99<frag> \n\n		

Table 5-8: Current Minimal Patterns and Seeds for Level 3 generalization.

For the level 3 generalization, the seeds and current minimal patterns are listed in Table 5-8. Each of the Level 3 seeds is more specific than the current minimal patterns and so none of the seeds are generalized further and since no negative training examples are matched, the generalization halts for the current positive training example and its minimal patterns are added to the list of minimal patterns for the text class. The current minimal patterns only match the current positive training example and so the second positive training example remains in the list of examples to be further generalized.

Generalization in the outer loop continues by generalizing the second positive training example. The initial seed for the level 0 generalization “<frag>\$37.25<frag>” does not match any negative training examples, but when it is generalized to “<frag>\$<dg><dg>.<dg><dg><frag>” or “<frag><DollarAmount><frag>”, both match negative training examples and so generalization continues to level 1.

The level 1 generalization starts with “<frag>\$37.25<frag>” as the current minimal pattern and “ <frag>\$37.25<frag>” and “<frag>\$37.25<frag> ” as seeds. The current minimal pattern is more general than either of these seeds, but because it consists entirely of literal pattern categories and the seeds do not match any negative training examples, the seeds will be further generalized. With these two seeds, each of their possible generalizations either match negative training examples or are more specific than the current minimal pattern: “<ws><frag>\$37.25<frag>”, “ <frag>\$<dg><dg>.<dg><dg><frag>”, “ <frag><<DollarAmount>><frag>”, “<frag>\$37.25<frag><ws>”, “<frag>\$<dg><dg>.<dg><dg><frag> ” and “<frag><<DollarAmount>><frag> ”.

In each of the next few levels, the current minimal pattern remains “<frag>\$37.25<frag>” because it continues to be more general than the initial seeds and further generalization of these seeds results in negative training example matches. Finally, in the level 9 generalization, the pattern “t Price: <frag>\$37.25<frag>” is generalized to “t Price: <frag>\$<dg><dg>.<dg><dg><frag>”, then “t Price: <frag>\$<dg>+.<dg>+<frag>”, then “t Price: <frag><<DollarAmount>><frag>” and finally to “<ws>Price:<ws><frag><<DollarAmount>><frag>”. None of these patterns match negative training examples and are therefore allowed to be further generalized. They also match more positive training examples than “<frag>\$37.25<frag>” and so replace it in the list of minimal patterns. The final pattern becomes a minimal pattern because if it is further generalized to “<lt><ws><lt><lt><lt><lt><lt><lt>:<ws><frag><<DollarAmount>><frag>” it matches negative training examples #1 and #2.

Generalization continues to level 10 because negative training examples were matched in level 9, but because of our current minimal patterns, each of the seeds for this level are found to be more specific and are not generalized further and

therefore no negative training examples are matched and generalization stops for this positive training example.

The minimal pattern is returned to the outer loop which matches the positive training example and removes it from the list of positive training examples to generalize. The minimal pattern “t<ws>Price:<ws><frag><<DollarAmount>><frag>” is then added to the list of minimal patterns for the text class because it is found to be neither more specific nor more general than either of the existing minimal patterns: “<frag>\$<dg><dg><dg>.<dg><dg><frag><ws>” and “:<ws><frag>\$<dg><dg><dg>.<dg><dg><frag>”.

Minimal Patterns for the “List Price” Text Class
<frag>\$<dg><dg><dg>.<dg><dg><frag><ws>
:<ws><frag>\$<dg><dg><dg>.<dg><dg><frag>
t<ws>Price:<ws><frag><<DollarAmount>><frag>

Table 5-9: The minimal patterns for the “List Price” text class.

CHAPTER 6 – THE TIBE CLASSIFIER

In order for TIBE to be able to highlight or extract text from web pages, it needs to generate a classifier that can match the appropriate text. In order to do this, TIBE generates regular expressions that act as the classifier for the current text class. To reduce the complexity of the training algorithm, the regular expressions generated by TIBE only use a small subset of the patterns and expressions possible in the regular expression grammar. When a classifier for a text class is generated, more than one regular expression may result; therefore a consumer of the matches must combine the matches from each individual regular expression and eliminate the duplicate matches between them.

6.1 Classifying Text

Given positive and negative training examples, a simple classifier could be generated that would match all of the positive examples and none of the negative examples. This simple classifier would be just the positive training examples that were selected by the user. For example if the user was training “Dollar Amounts” and had added “\$9.99” and “\$12.37” as positive examples, the classifier could just match all instances of “\$9.99” and “\$12.37” within the document. This would not be very interesting and would not be very useful to the user. They would need to select every instance of the text class in all documents that they wanted to have it highlighted in.

So instead of matching the literal text of the positive training examples we would like a classifier that could match additional instances of the text class without those instances having to be specifically added as positive training examples. Regular expressions do really well at matching text for a given pattern and so can be used to do matching for the classifier. The TIBE classifier is

essentially creating a regular expression to do the matching rather than the harder way of creating the regular expression by hand.

Regular expressions are essentially a programming language for matching text and support many operations and matching patterns. The TIBE classifier only generates regular expressions that have a small subset of the operations and patterns in the regular expression language. This simplifies the matching process and the generation of a regular expression classifier from the TIBE training algorithm. Table 6-1 lists the regular expression patterns that could be included in a regular expression generated by a TIBE classifier.

Character Class	Regular Expression Pattern
Literal Character (ie "a", "b", "c", "1", "2", "3")	Literal Character (ie "a", "b", "c", "1", "2", "3")
Any Letter	[A-Za-z]
Any Digit	\d
Any Whitespace	\s
Multiple	[A-Za-z]+, \d+, \s+
Zero-Width Positive Look-Behind Assertion	(?<=)
Zero-Width Positive Look-Ahead Assertion	(?=)

Table 6-1: The regular expression matching patterns that the TIBE classifier uses in its generated regular expressions.

The Zero-Width Positive Look-Behind Assertion and the Zero-Width Positive Look-Ahead Assertion aid in matching the pre-text and post-text respectively of a classifier. The pre-text or post-text regular expressions are placed in one of those assertions to add additional contextual constraints to the text to be matched without being included in the match. This is an important part to the TIBE classifier because often the pre-text and/or the post-text parts of

the classifier help differentiate one text class from another. For example birth dates and death dates both have dates within them, but birth dates tend to be in the form “Birth: {date}” or “Birth Date: {date}” where death dates look like “Death: {date}” or “Death Date: {date}”. The many prices on a page on Amazon.com is another example where the pre-text and post-text are important to the TIBE classifier. A list price might look like “\$9.99”, a sales price like “\$9.99” and a shipping amount like “\$9.99”. The only way to not match a salesprice and a shipping amount when searching for a list price is to include some of the pre-text in the regular expression (ie “(?<=('listprice'>))(\$\d\.\d\d)”).

6.2 Classifying/Recognizing Engine

More than one regular expression can be generated by the TIBE trainer for the classifier from a given set of positive and negative examples. Because of this, the consumer of the regular expressions generated by the TIBE classifier must combine the matches of each regular expression and eliminate duplicate match locations. For example, if the TIBE trainer produced a classifier that included both “Death:<dg>+</dg>+</dg>+” and “<dg>+</dg>+</dg>+<ws>” as minimal patterns, the resulting regular expressions would both match “Death:12/14/2005 A.D.”. The duplicate match would need to be filtered out so that each match in the resulting list of matches was unique.

CHAPTER 7 – EVALUATION

7.1 Introduction

In order to evaluate the TIBE training algorithm, we developed a program that would allow us to test how well the algorithm converged on a classifier that would match the text class being trained. The training algorithm was tested on a variety of different web pages using text classes associated with each type of page. The first three pages test the algorithm on pages with computer-generated content: Amazon.com Book Pages, Amazon.com Camera Pages and Amazon.com Book search list pages. The last two pages test the algorithm on pages with human-generated content: Obituaries and Utah Genealogical Search.

7.2 TIBE Analyzer

In order to evaluate the training algorithm, it needs to be run on a golden standard of marked-up text. All of the text for each of the text classes being trained is annotated so that the analyzer can select one of them as a positive training example and can verify whether the classifier has matched all examples of the text class and not matched any text that is not part of the text class.

Through our analyzer we are able to load the raw HTML of web pages and annotate it for the text classes related to the page. Then from the same interface, a text class could be analyzed against a specific page or a set of pages of the same type. The analyzer would start by adding one of the examples of the text class as a positive training example which would cause the algorithm to create a classifier. The classifier would then be used to find all matches within the text and the number of positive and negative matches would be recorded. If there were any positive examples that had not been matched by the classifier, then one of the missing examples would be added in the next round as another positive training example. If there were any negative matches amongst the classifier's

matches, one of them would be added as a negative training example for the next round.

Once the positive and negative training examples had been added, the analyzer proceeds to retrain its classifier and re-classify the text. The analyzer continues this cycle until all positive text class examples have been matched and the classifier does not match any non-text-class examples. During this process, several metrics are measured along the way in order to be able to evaluate the performance of the algorithm.

7.3 Computer Generated Webpage Content

The first sets of web pages that we analyze are web pages that are primarily computer-generated content. There is some human generated content on these pages, but we are not training any text that would be contained within them. Computer generated content is different from human-generated content in that it is usually much more uniform and often has very identifiable pre-text and post-text. This especially occurs on web pages as content is often times put into tables or has markup surrounding it that indicates how it should be displayed.

The computer generated content that we are analyzing has come from 3 different locations on the Amazon.com website. The first type of web page is Amazon Book pages. These are the pages that a user would be directed to in order to find out more detailed information about a book. The next type of web page is Amazon Camera pages. These are similar to the Amazon Book pages in that they show detailed information about specific cameras that Amazon has for sale. They are different from the Amazon Book pages because they have some different types of information that can be trained. The third type of web page being analyzed is the Amazon Book Search pages. These are the search list page that one might encounter on Amazon.com when searching for different types of

books. Instead of giving detail around specific books, each page has a summary of several books on the same page.

There were several text classes that we trained on the computer generated web pages. Table 7-1 lists the text classes and short descriptions for each.

Text Class	Related To	Description
Dollar Amount		A decimal amount preceded by a \$ symbol.
List Price	Dollar Amount	A "Dollar Amount" that indicates the regular price of products on the Amazon.com web pages.
Sales Price	Dollar Amount	A "Dollar Amount" that indicates the on-sale price of products on the Amazon.com web pages.
Savings	Dollar Amount	A "Dollar Amount" that indicates amount saved off of the list price of the product.
Buy New	Dollar Amount	A "Dollar Amount" indicating how much the product costs new.
Used & New	Dollar Amount	A "Dollar Amount" indicating how much the product is selling for in the third-party market.
Date		A date of the form "January 31, 2007".
First Available Date	Date	The "Date" when the product was first available on Amazon.com.
ISBN-10		A 10 digit ISBN of the form "156222105X" or "0439139597".
ISBN-13		A 13 digit ISBN of the form "978-0439139595".
Weight		How much the product weighs.
Sales Rank		Indicates the popularity of a product in its same category within Amazon.com.
ASIN		The Amazon Standard Identification Number and is 10 characters that can be numbers or letters.
Model Number		The model number of a product.
Shipping Weight		The weight of the product when being shipped.
Product Dimensions		The height, width and length of the product given in a Length x Width x Height notation.
Book Binding		A string indicating what the book binding is (ie Paperback, Hardbound, etc)

Table 7-1: Text classes for computer generated content.

7.3.1 Summary of Results

The TIBE training algorithm did well against the computer generated content. The time to generate the classifier once all the positive and negative training examples had been chosen averaged 1.89 seconds with 64% of the trainings taking less than 1.5s. There is still much that could be done to optimize the inner workings of the training algorithm but these numbers indicate that the algorithm itself reduces much of the possible work.

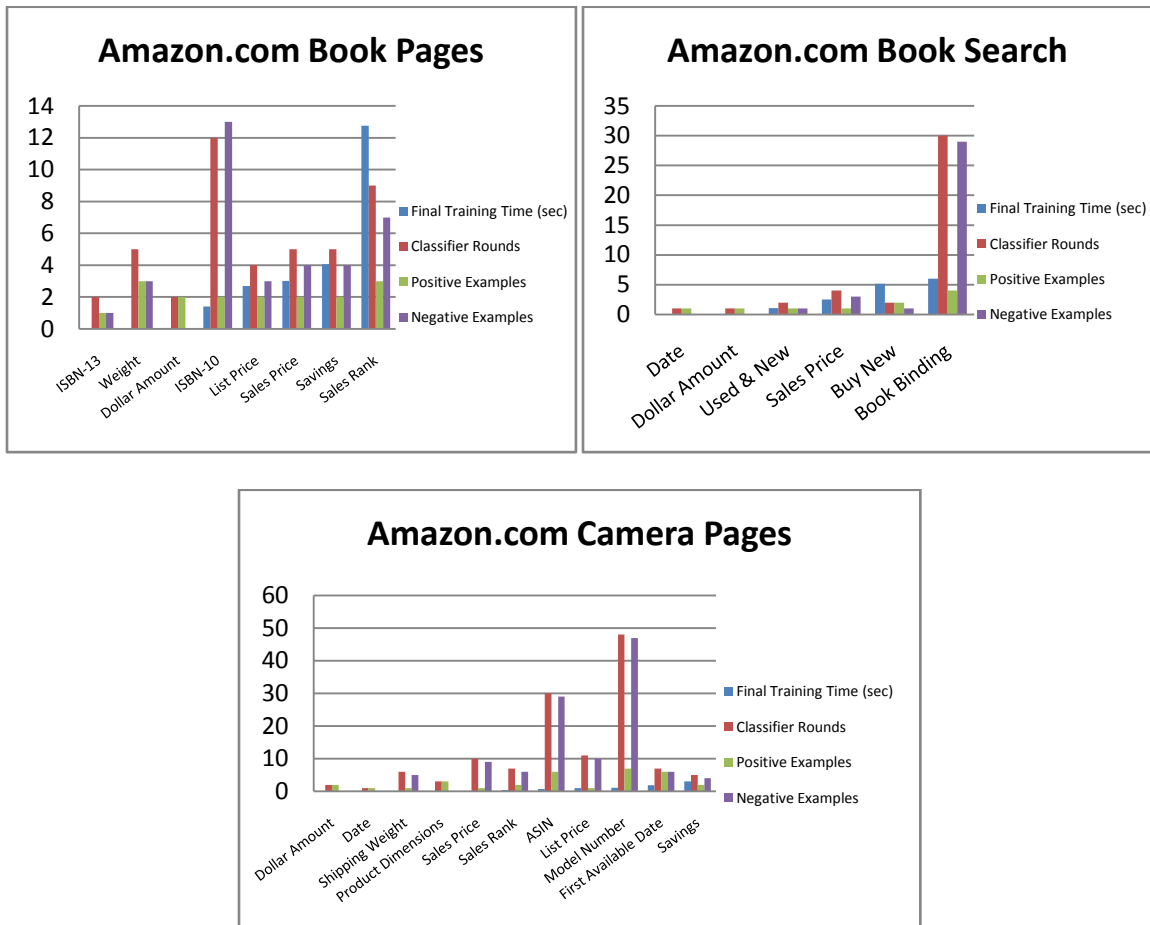


Figure 7-1: Graphical Summary of the training results for computer-generated content.

Figure 7-1 shows the results of the training using four metrics. The “Final Classifier Time” measures the time to generate the classifier once all of the necessary positive and negative training examples had been selected. The

“Classifier Rounds” indicates the number of rounds of choosing positive and/or negative training examples it took to fully train the classifier. The “Positive Examples” and “Negative Examples” show how many positive and negative training examples were necessary in order to fully train the classifier.

The number of rounds that it took to train each text class averaged around 8 rounds with 64% of the trainings requiring 6 or less rounds to fully train the classifier. An end-user is going to want to be able to classify a text class in as few rounds as possible. 64% requiring 6 or less rounds is not as high as we would have liked, but it is still a majority.

On average, 2.36 positive training examples and 7.4 negative training examples were needed to train each text class. 88% of the text classes required 4 or fewer positive training examples and 60% of them required 4 or fewer negative training examples. For many simple text classes, only one positive training example would be needed if it was picked to be fairly representative of its text class. Because the TIBE analyzer just picked a random positive text example, sometimes it did not choose one that represented its text class well. For example it might pick a dollar amount of “\$8.37” which would generalize to “\$<dg>.<dg>+”, but a better positive training example would be “\$37.56” which would generalize to “\$<dg>+.<dg>+”. More negative training examples were needed for these text classes because they were necessary in order to make the minimal patterns for the text class more specific so that they would only match their patterns.

Several of the text classes took more than ten rounds to train. These text classes include the ISBN-10, Sales Rank, Book Binding, ASIN and Model Number. One can see from the graphs that these text classes also required several negative training examples to train. Most of these text classes have fairly generic selected text. The selected text is either all numbers or all letters. When

the positive examples were generalized, they ended up matching many selections of text that were not part of the current text class and therefore required more negative examples in order to make the pre-text and/or post-text more specific.

7.4 Human-Generated Webpage Content

Compared to computer-generated web content, human-generated web content is much harder to train because of the variability of how humans enter information on a web page. For example, if a computer is outputting dates on a web page, it has pre-defined formats and often times will use a single format for all dates on a set of web pages. Humans are much less predictable. One human may type his dates in the format "January 2, 2004" while another human may use the format "1/2/2004". What makes it even more complicated is that some dates may be formatted like "January 2,2004", "Jan. 2, 2004" or "Jan.2, 2004". We would want to train each of these as a date, but each would probably need to be a separate positive training example because it differs slightly from the other dates. In our evaluation, we train fewer text classes per set of web pages because training the text classes is complex enough to offer a good evaluation of the TIBE trainer.

The good news for the TIBE training algorithm is that it was able to train a classifier for all of the text classes that we tested against in the two types of human-generated web pages that we evaluated (although the minimal patterns were not always optimal). The first set of web pages are a collection of 8 obituaries and the second set of web pages are a collection of 8 pages of entries for a Utah genealogical query. Table 7-2 lists the text classes trained along with descriptions of each.

Text Class	Related To	Description
Date		A date in several formats (MM/DD/YYYY, DD MMM YYYY, MMM. DD, YYYY, etc)
Birth Date	Date	A "Date" that represents a birth date.
Death Date	Date	A "Date" that represents a death date.
Date Range		A range of years in the form 1977 ~ 2007.
Age		A number indicating the number of years a person was alive.

Table 7-2: Text classes for human generated content.

7.4.1 Summary of Results

Figure 7-2 shows the results of the training using the same four metrics measured with the computer-generated content.

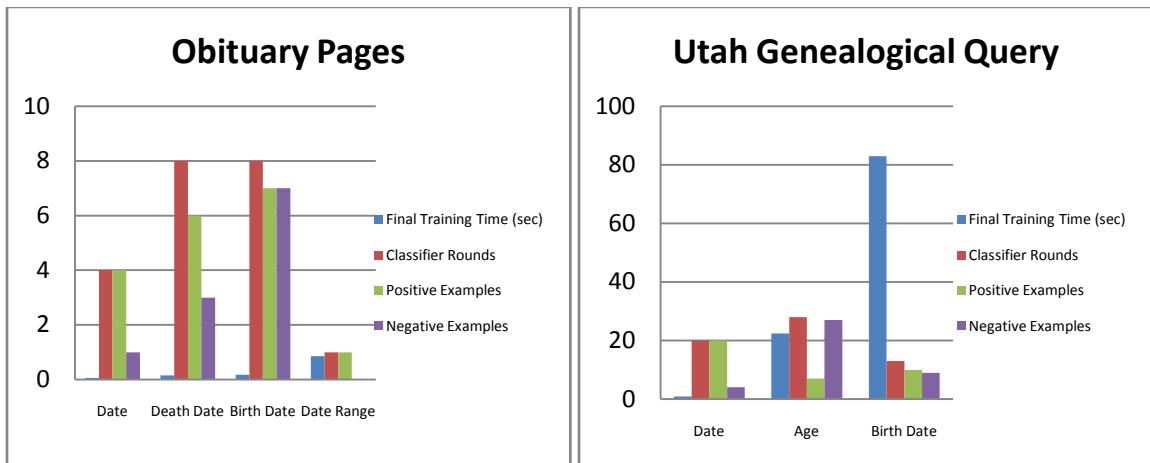


Figure 7-2: Graphical Summary of the training results for human-generated content.

Once the text classes were fully trained, 71% of them trained in less than 1 second. Despite this though, the average training time was 15.35 seconds because the Age text class took 22 seconds to train and Birth Date text class took 83 seconds to train on the Utah Genealogical Query pages. For Age, this is probably because there are not many identifying features for numbers that are ages so therefore it required several positive training examples and many more negative training examples. For Birth Date, most of the time was spent

generalizing two of the positive training examples. One of the positive training examples had to test 7,010 patterns and the other had to test 12,113 patterns. This is probably due to the birth date text class needing to use much of its pre-text and post-text in order to properly classify it.

The number of rounds to fully classify the text classes averaged 12 rounds with 57% of the text classes needing 8 rounds or less to train. The average is again skewed by two of the text classes that required more than 20 training examples to train and so required more rounds to train. Considering that the human generated content varies in format much more than the computer generated content, these numbers show fairly good performance on human-generated content.

In order to train the human generated text classes, an average of 7.86 positive training examples were needed and an average of 7.28 negative training examples were needed. 71% of the text classes needed 7 or fewer positive and negative training examples in order to be fully trained. The high positive training example count can be attributed to the variability of human-generated content (especially with dates which is one of the main text classes trained). The number of negative training examples required to train is actually quite low compared to the positive training examples. This again could be attributed to the variability of the human generated content. Because the human generated content varies so much, items within one text class are not likely to be similar enough to other text outside that text class to cause the classifier to misclassify it.

7.5 Conclusions

Although there certainly is room for improvement, the TIBE training algorithm has done fairly well in most cases in training the text classes tested. In the majority of text classes, it classifies in less than 1.5 seconds, fully classifies in 8

rounds or less, and needs fewer than 8 positive and negative training examples. It performed very well with the computer generated content and performed decently with the human generated content. In all cases, it was able to eventually derive a classifier that would fully classify the text class.

CHAPTER 8 – SUMMARY

8.1 What was the problem?

There is a lot of information that can be harvested from web sites on the internet, but manually extracting this information is tedious, slow and error prone. Users want to be able to automate the extraction of this information and there are tools like regular expressions and conceptual-modeling that help automate this extraction of information. The problem with these methods of extracting information is that regular expressions are hard to write and have a high learning-curve and conceptual-modeling ultimately uses regular expressions in order to extract information from web pages. This means that either the end-user must learn regular expressions or use a program with a set of regular expressions pre-programmed. The first option is not realistic for most computer users and the second option is very limiting because web pages are very diverse in their ways of representing information. End users need an easy way to extract information from web pages that would not require them to learn regular expressions or be limited by a pre-programmed set of regular expressions.

8.2 The TIBE Solution

TIBE solves this problem by making it possible for an end-user to train the computer to recognize the text that he is trying to extract. This is accomplished through an easy-to-use user-interface that allows the user to quickly train a classifier by selecting positive and negative training examples and giving the user instant feedback on the quality of the classifier by highlighting the recognized text after each training example is added. The TIBE training algorithm is built to be able to take the positive and negative training examples provided by the end-user and generate a classifier that will be able to extract the

positive training examples as well as other text that would also fit into that text class.

8.3 What remains to be done?

Although the TIBE training algorithm was able to ultimately derive a set of minimal patterns that would classify all examples of the text class in each of our evaluations, there are several things that we feel could be tried to optimize the number of minimal patterns generated, the time to generate them and the number of rounds to fully train a classifier. These are all things that as we were evaluating the algorithm, we noticed might help the algorithm to be more efficient, but did not have time to implement and evaluate.

During each round of the evaluation, the generalization of the positive training examples would essentially start over. This was because another positive or negative training example was added and might affect the outcome of the generalization of the previous positive training examples. One optimization here might be to save the information on the previous run and instead of re-generalizing the positive training examples use some of the previously known information to make the previous minimal patterns more specific so that they don't match the negative training examples any more.

For the text classes that we were training, the first round of generalization generalizes to the most general form of the selected-text. For a date this might be "<dg>+<dg>+<dg>+" but for a year it would be "<dg>+" and for any one word text class, it would be "<1t>+". The last two minimal patterns are not likely to be the classifiers that one is trying to train and the first would match any number on the web page and the second one would match every word on the web page. This creates a lot of negative matches which would overwhelm the web page as every word would be highlighted. One proposal to fix this would be to have some

permanent negative training examples like “<dg>+” and “<lt>+” that would prevent these patterns from being considered for the minimal patterns.

As can be seen in several of the evaluations that we did in chapter 7, often times several minimal patterns are generated from one positive training example. Some of these minimal patterns are obviously more optimal and more likely to match other examples of the text class. One of the optimizations that could be evaluated would be to figure out metrics to rate how general a pattern is and to drop some of the patterns that are the most specific of the bunch. Several things would probably need to be looked at here. What is the “specific” threshold at which a pattern should be dropped and how does that affect the evaluation of the text class? Does it increase the number of rounds to fully train the classifier because we are not matching all of the annotated examples of the text class?

In the TIBE training algorithm, we split training examples into a pre-text, selected-text and post-text and generalized each of these parts of the training examples separately. One of the problems that this presented to the training is when the pre-text or post-text of a training example is the same amongst the positive training examples, but the selected-text was slightly different. This could be like an ISBN number that has different combinations of letters and numbers, or it might be like the book binding where some of the selected-text had three words but others only had one word. Instead of generalizing the pre-text, post-text and selected-text, another possibility would be to generalize the starting position and the ending position of the training examples. This should make it so that if the selected-text is similar across positive training examples, it would be included in the minimal patterns and not included if it is not similar across training examples.

One thing that we noticed during the evaluations was that our logic for giving more weight to patterns that matched more positive training examples often times did not get used and therefore the minimal patterns were frequently not as optimal or general as they should have been. This was because once a positive training example had been added, the classifier usually matched all other similar text fragments and so did not need to add another positive training example that would also be matched by the minimal patterns to give them more weight. In order to help this part of the algorithm, we might try adding text fragments that were matched by the classifier to a list of pseudo-positive training examples. Patterns that match more items in this list would get more weight than patterns that match fewer items while training.

As was mentioned a few times in the evaluation, there were times when it appeared that we might be able to get a better set of minimal patterns if we allowed the algorithm to further generalize patterns that are found to be more specific than existing minimal patterns. For example, “<ws><frag>12/31/2006<frag>” is more specific than “<ws><frag>12/31/2006<frag>”, but if we let it be generalized further, “<ws><frag><<date>><frag>” might match more positive training examples and is no longer more specific than “<ws><frag>12/31/2006<frag>”. The only problem with generalizing further is that it might mean that we will be testing many more patterns as minimal patterns and there are many cases where generalizing further will not result in a better minimal pattern and so we will want to figure out if there are ways to determine whether we should generalize further or not. This might depend on how general or specific the current minimal patterns are.

The last thing that could be done to optimize the training algorithm might be to store the generalization tree for generalizing the selected-text since currently it is re-generalized on each level several times as it is tested with different

combinations of the pre-text and post-text. By saving the generalization tree, we would be able to save some of the time spent applying the different generalization rules against each of the patterns.

There is still much that could be done to help optimize the TIBE training algorithm, but this thesis has shown that it can be done and with more research and development, it could be made into a feature that could be included in web browsers to aid regular users in their text extraction needs.

BIBLIOGRAPHY

- [Cre01] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. Technical Report n. RT-DIA-64-2001, D.I.A., Universit a di Roma Tre, 2001.
<http://citeseer.ist.psu.edu/crescenzi01roadrunner.html>
- [Emb98A] D. W. Embley, D. M. Campbell, R. D. Smith, S. W. Liddle. Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents. In *CIKM'98 Proceedings*. 1998.
<http://www.deg.byu.edu/papers/cikm98.pdf>
- [Emb98B] D.W. Embley, D.M. Campbell, Y.S. Jiang, Y.-K. Ng, R.D. Smith, S.W. Liddle, and D.W. Quass. A conceptual-modeling approach to extracting data from the web. In *Proceedings of the 17th International Conference on Conceptual Modeling (ER'98)*, Singapore, November 1998. <http://citeseer.ist.psu.edu/24307.html>
- [Emb99] D.W. Embley, D.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.-K. Ng, and R.D. Smith. Conceptual-model-based data extraction from multiple-record web pages. In *Data & Knowledge Engineering*, 1999.
<http://citeseer.ist.psu.edu/embley99conceptualmodelbased.html>
- [Emb04A] D.W. Embly, and T. Walker. Automating the Extraction of Genealogical Information from the Web. In *Fourth Annual Workshop on Technology for Family History and Genealogical Research*, March 2004.
<http://www.deg.byu.edu/papers/FHT04.pdf>
- [Emb04B] D.W. Embly. Toward Semantic Understanding – An Approach Based on Information Extraction Ontologies. *The Fifteenth Australasian Database Conference*, January 2004. <http://www.deg.byu.edu/papers/KeynotePaper.ADC04.pdf>
- [Mil99] R.C. Miller and B.A. Myers. Lightweight Structured Text Processing. In *Usenix Annual Technical Conference*. 1999. Monterey, California: pp. 131-144.
<http://citeseer.ist.psu.edu/miller99lightweight.html>
- [Mil01] R.C. Miller and B.A. Myers. Interactive simultaneous editing of multiple text regions. In *Proc. USENIX Tech. Conf.*, pp 161-174, June 2001.
<http://citeseer.ist.psu.edu/miller01interactive.html>

- [Mil02] R. Miller and A. Myers. Multiple Selections in Smart Text Editing. In *Proceedings of IUI 2002*, San Francisco, CA, pp. 103-110, January 2002.
<http://citeseer.ist.psu.edu/miller02multiple.html>
- [Mit97] T. M. Mitchell. Chapter 2: Concept Learning and the General-To-Specific Ordering. In *Machine Learning*. pp. 20-51 New York: The McGraw-Hill Companies, Inc, 1997.
- [Wit93] I. H. Witten and D. Mo. TELS: Learning Text Editing Tasks from Examples. In *Watch What I Do: Programming by Demonstration*. pp. 183-203, Cambridge, MA: The MIT Press, 1993.