A Semi-supervised Algorithm for Pattern Discovery in Information Extraction from Textual Data

Tianhao Wu and William M. Pottenger

Computer Science and Engineering, Lehigh University
{tiw2, billp}@lehigh.edu

Abstract. In this article we present a semi-supervised algorithm for pattern discovery in information extraction from textual data. The patterns that are discovered take the form of regular expressions that generate regular languages. We term our approach 'semi-supervised' because it requires significantly less effort to develop a training set than other approaches. From the training data our algorithm automatically generates regular expressions that can be used on previously unseen data for information extraction. Our experiments show that the algorithm has good testing performance on many features that are important in the fight against terrorism.

1 Introduction

Criminal Justice is an important application domain of data mining. There are thousands of incident reports generated daily around the world, many of them in narrative (unstructured) textual form. Information extraction techniques can be used to identify relational data in such unstructured text, which in turn is used in a variety of computational knowledge management applications such as link analysis.

After studying hundreds of incident reports, we find that regular expressions can be readily employed to express patterns of features. For example, a suspect's height might be recorded as "{CD} feet {CD} inches tall", where {CD} is the part of speech tag for a numeric value. We have developed a semi-supervised algorithm for automatic discovery of regular expressions of this nature.

We term our approach 'semi-supervised' because it requires significantly less effort to develop a training set than other approaches. Instead of labeling the exact location of features in a training set, the training-set developer need only record whether a specific feature of interest occurs in a sentence segment. For instance, if a segment is "A 28 year old woman was walking from the store to her car.", and the feature of interest is Age, then the training-set developer need only assign this segment the label Age. Using this training data, our algorithm discovers a regular expression for a person's age. For example, "{CD} year old" might be found as a regular expression[1] for a

[1] Currently, only words and English part of speech tags are used in the discovery process.

person's age in this particular example. The automatically generated regular expressions can be used to extract various features from previously unseen data. Our experiments show that the algorithm has good testing performance on many features that are important in homeland defense.

Although much work has been done in the field of information extraction, relatively little has focused on the automatic discovery of regular expressions. Stephen Soderland developed a supervised learning algorithm, WHISK [1], which uses regular expressions as patterns to extract features from semi-structured and narrative text. Eric Brill [2] applied his transformation-based learning framework to learn reduced regular expressions for correction of grammatical errors in text. A crucial difference between these two approaches and ours is that WHISK and Brill's approach require the user to identify the precise location of features for labeling while our approach requires only that instances (segments) be labeled. Moreover, our regular expressions are more general since our approach supports the inclusion of the logical "OR" operator in regular expressions, while Brill's approach does not.

Michael Chau, Jennifer J. Xu, and Hsinchun Chen have published results of research on extracting entities from narrative police reports [5]. They employed a neural network to extract names, addresses, narcotic drugs, and items of personal property. Their cross-validation accuracy results vary from a low of 46.8% to a high of 85.4%. In our approach, however, we achieve significantly better results without limiting ourselves to noun phrases. In addition, we are able to extract a greater variety of textual features that can be used, for example, in link analysis of modus operandi.

The article is organized as follows. In section 2, we provide a framework for understanding our algorithm. Our approach is described in section 3, and in section 4 we detail preliminary experimental results. Finally, we present our conclusions, discuss future work, and give our acknowledgements in section 5.

2 Definitions

In this section, we start with the standard definition of a regular expression, and then define a reduced regular expression as used in our algorithm.

Regular expression: "Given a finite alphabet Σ, the set of regular expressions over that alphabet is defined as:

$\forall a \in \Sigma$, a is a regular expression and denotes the set {a}.

if r and s are regular expressions denoting the languages R and S, respectively, then (r+s), (rs), and (r*) are regular expressions that denote the sets $R \cup S$, RS and R* respectively." [2] [7]

Reduced regular expression (RRE): Our reduced regular expression is at first glance similar to that defined in [2]. However, there are some significant differences. Given a finite alphabet Σ, our reduced regular expression is defined as a set:

$\forall a \in \Sigma$, a is a RRE and denotes the set {a}.

~a* is a RRE and denotes the Kleene closure of the set $\Sigma - a$.

(^, $, \s \S \w \W) $\subset \Sigma$, where ^ is the start of a line, $ is the end of a line, \s is any white space, \S is any character except white space, \w is any alphanumeric character, and \W is any non-alphanumeric character.

(\w)* is a RRE denoting the Kleene closure of the set {\w}.

(\w){i,j} is a RRE denoting that \w is repeated between i and j times, where $0 \leq i \leq j$.

a? is a RRE and denotes that a is an optional part of the RRE.

if r and s are RREs denoting the languages R and S, respectively, then (r+s) and (rs) are RREs that denote the sets $R \cup S$ and RS, respectively.

Some examples of regular expressions that are not RREs are: "a*", "(ab)*", and "a+". We have not found it necessary to support these regular expressions to achieve high accuracies.

3 Approach

In this section we present our approach to the discovery of RREs from a small set of labeled training segments. The process begins with the processing of datasets. Next, a greedy algorithm is applied. Finally, RREs for the same feature are combined to form a single RRE.

Pre-Processing Pre-processing includes segmentation, feature identification, segment labeling, and part of speech tagging. Each incident report is split into segments at this stage and becomes an instance in our system. We assume that no features cross segments. This assumption is practical for a number of important features, including those listed in Table 1. A domain expert must identify features that will be extracted such as 'Height', 'Weight', 'Eye Color', etc. Each segment is then assigned labels that correspond to the set of features present. After labeling, each feature has a *true set* corresponding to true positive segments and *false set* corresponding to true negative segments. Finally, each word is (automatically) assigned its part of speech [3].

Learning Reduced Regular Expressions The goal of our algorithm is to discover sequences of words and/or part of speech tags that, for a given feature, have high frequency in the true set of segments and low frequency in the false set. The algorithm first discovers the most common element of an RRE, termed the *root* of the RRE. The algorithm then extends the 'length' of the RRE in an "AND" learning process. During the "OR" learning process, the 'width' of the RRE is extended. Next, optional elements are discovered during the "Optional" learning process. The algorithm then proceeds with the "NOT" learning process, and finally discovers the start and the end of the RRE. Figure 1 depicts the entire learning process.

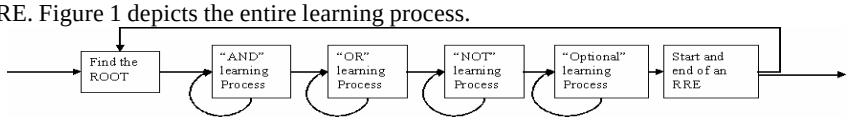

Figure 1: RRE Discovery Process

Our approach employs a covering algorithm. After one RRE is generated, the algorithm removes all segments covered by the RRE from the true set. The remaining segments become a new true set and the steps in Figure 1 repeat. The learning process stops when the number of segments left in the true set is less than or equal to a threshold d. We describe the details of the first three steps of the algorithm in what follows. [6] describes the details of the other steps.

To discover the root of a RRE, the algorithm matches each word and/or part of speech tag (specified as a simple RRE) in each true set segment against all segments. The performance of each such RRE in terms of the metric $F_b = ((b^2+1)PR)/(b^2P+R)$ (from [4]) is considered. In this formula, $P = precision = TP/(TP+FP)$ and $R = recall = TP/(TP+FN)$, where TP are true positives, FP are false positives, and FN are false negatives. The parameter β enables us to place greater or lesser emphasis on precision, depending on our needs. The word or part of speech tag with the highest score is chosen as the root of the RRE. In other words, the algorithm discovers the word or part of speech tag that has a high frequency of occurrence in segments with the desired feature (the true set). Meanwhile, it must also have low frequency in segments that do not contain the desired feature (the false set).

After the root is discovered, the algorithm places new candidate elements immediately before and after the root, thereby forming two new RREs. Any word or part of speech tag (other than the root itself) can be used in this step. The RRE with the highest score will replace the previous RRE. The new RRE is then extended in the same way. Adjacency implies the use of an "AND" operator. As before, candidate words and parts of speech are inserted into all possible positions in the RRE. The algorithm measures the performance of each new RRE and the one with the highest score is selected if its score is greater than or equal to the previous best score. In this sense our algorithm is greedy. The RRE learned after this step is termed R_{AND}. The overall complexity of this step is $O(N^2)$, where N is the number of elements in R_{AND} [6].

After the "AND" learning process is complete, the algorithm extends the RRE with words and part of speech tags using the "OR" operator. For each element in R_{AND}, the algorithm uses the "OR" operator to combine it with other words and/or part of speech tags. If the newly discovered RRE has a better F-measure than the previous RRE, the new RRE will replace the old one. The complexity of "OR" learning process is O(N).

The "Optional" learning process and the "NOT" learning process are then applied. Each of them has complexity O(N). Finally the algorithm discovers the start and the end of the current RRE. For details on these three steps, please refer to [6].

Post Processing During each iteration in Figure 1, one RRE is generated. This RRE is considered a sub-pattern of the current feature. After all RREs have been discovered for the current feature (i.e., all segments labeled by the feature are covered), the system uses the "OR" operator to combine the RREs.

In this section we have described a greedy covering algorithm that discovers a RRE for a specific feature in narrative text. In the following section we present our experimental results.

4 Experimental Results

In this section, we describe the datasets for training and testing. We use domain expert labeled segments for training. We employ two different methods to evaluate the training results. The first method tests whether segment labels are correctly predicted. We term this segment evaluation. The second method evaluates the performance of the model with respect to an exact match of the feature of interest. The metric F_b (with β=1 to balance precision and recall) is used to evaluate the test performance with both methods. We use the widely employed technique of 10-fold cross-validation to evaluate our models.

Our training set consists of 100 incident reports obtained from Fairfax County, USA. These reports were automatically segmented into 1404 segments. The first column of Table 1 depicts 10 features supported by our system. *Eye Color* and *Hair Color* are not well represented in our dataset due to their infrequent appearance in the Fairfax County data.

The result of the training process is one RRE for each feature. For example, "(CD (NN)? old)|(in IN (MALE)? CDS)" is a high-level abstraction of the RRE for the feature *Age*. In this example, "NN" is the part of speech tag for noun, "MALE" is the tag for words of male gender such as "his", and "CDS" is the tag for the plural form of a number (e.g., "twenties" or "teens").

After completing 10-fold cross-validation, there are 10 test results for each feature. The average precision, recall and F_b (β=1) for these ten results is depicted in Tables 1 and 2. Table 1 contains the results based on segment evaluation, and Table 2 depicts the result of testing for an exact match. We also include a column for the average absolute number of true positives covered by each RRE.

Table 1. 10-fold cross-validation test performance based on segment evaluation

Feature	Precision	Recall	F_b	Avg. TP
Age	97.27%	92.38%	94.34%	13
Date	100%	94.69%	97.27%	8.8
Time	100%	96.9%	98.32%	8.9
Eye Color	100%	100%	100%	1
Gender	100%	100%	100%	33.6
Hair Color	60%	60%	60%	0.8
Height	100%	98%	98.89%	2.4
Race	95%	96.67%	94.67%	3.3
Weekday	100%	100%	100%	9.8
Weight	90%	90%	90%	1.9

In Table 1, *Eye Color*, *Gender* and *Weekday* have perfect test performance (100%) in part because we have modified the lexicon used in part of speech tagging to label these features during pre-processing. The performance of *Age*, *Date*, *Time*, *Height*, *Race*, and *Weight* are also excellent (F_b =90%). Although we also modified the lexicon to include a special "month" tag, which is a part of *Date*, the performance of *Date* is not perfect. This is a result of the fact that "2001" and "2002" cover over 95% of the

years in our dataset, so the algorithm discovers "2001" or "2002" as a sub-pattern in *Date*. As a result, years other than "2001" and "2002" were not recognized as such during testing. This caused a slight drop in performance for the *Date* feature.

In order to address this issue we developed a more complete training set and re-evaluated our algorithm on the *Date* feature. In the new training set, there were ten "2002", ten "2001", ten "2000", ten "1999" elements, as well as a few "none" year elements. In this case our algorithm discovered the RRE "^ ([0-9a-zA-Z]){4} CD $". This is a more general pattern for detection of the year in the *Date* feature.

The performance of *Hair Color* is, however, not as good. As noted, this is due to the lack of *Hair Color* segments in test sets (2, 4, 5, 8). However, the test performance on sets (1, 3, 6, 7, 9, 10) is perfect for *Hair Color* (F_b of 100%). Therefore, we conclude that the RRE discovered for the feature *Hair Color* is optimal.

In a practical application, a user is interested in the exact feature extracted from narrative text rather than the segment. Therefore, it is necessary to evaluate our RREs based on their ability to exactly match features of interest. An exact match is defined as follows: "If a sub-string extracted by an RRE is exactly the same as the string that would be identified by a domain expert, then the sub-string is an exact match." An example of an exact match is as follows: if a human expert labels "28 year old" as an exact match of the feature *Age* in *"A 28 year old woman was walking from the store to her car."*, and an RRE also discovers "28 year old" as an age, then "28 year old" is an exact match. The result of our experiments in exactly matching features of interest is depicted in Table 2.

Table 2. 10-fold cross-validation test performance based on exact match

Feature	Precision	Recall	F_b	Avg. TP
Age	92.61%	88%	89.83%	12.4
Date	100%	94.69%	97.27%	8.8
Time	87.87%	85.01%	86.32%	7.8
Eye Color	100%	100%	100%	1
Gender	100%	100%	100%	33.6
Hair Color	60%	60%	60%	0.8
Height	95%	93.5%	94.17%	2.2
Race	90%	91.67%	89.67%	3
Weekday	100%	100%	100%	9.8
Weight	82.5%	82.5%	82.5%	1.7

In the exact match results in Table 2, *Age, Time, Height, Race, and Weight* have slightly lower performance than in segment evaluation. Sometimes the string accepted by a RRE is a sub-string of the actual feature. For example, suppose that the correct value for a particular instance of the *Time* feature is "*from/IN 9/CD :/: 00/CD p/NN ./. m/NN ./. until/IN 3/CD :/: 00/CD a/DT ./. m/NN ./NN*", but the string accepted by the RRE is "from/IN 9/CD :/: 00/CD p/NN ./. m/NN ./.". In this case, the algorithm failed to match the feature exactly. Nevertheless, these features still have very good performance, with high precision and recall (both of them are greater than 80%). On the other hand, *Date, Gender, Eye Color, Hair Color, and Weekday* all have the same perform-

ance as that achieved during segment evaluation. In fact, it turns out that the RREs discovered automatically for these five features are exactly the same patterns developed manually by human experts who studied this same dataset. Based on these exact match results, we conclude that our approach to RRE discovery has good performance on all ten features supported in our current system.

5 Conclusion

We have presented a semi-supervised learning algorithm that automatically discovers Reduced Regular Expressions based on simple training sets. The RREs can be used to extract information from previously unseen narrative text with a high degree of accuracy as measured by a combination of precision and recall. Our experiments show that the algorithm works well on ten features that are often used in police incident reports.

One of the tasks ahead is to develop more sophisticated segment boundary detection techniques (similar to existing sentence boundary detection techniques). Other future work includes the application of our techniques to extract additional features of interest such as those used in describing modus operandi. We plan to focus on these two tasks in the coming months.

This work was funded by the Pennsylvania State Police (PSP) under a subcontract with the Lockheed-Martin Corporation. We gratefully acknowledge the Lockheed-Martin/PSP team, co-workers, and our families for their support. Finally, we gratefully acknowledge our Lord and Savior, Yeshua the Messiah (Jesus Christ), for His many answers to our prayers. Thank you, Jesus! ☺

References

[1] S. Soderland: Learning information extraction rules for semi-structured and free text. Machine Learning, 34(1-3):233-272. (1999)

[2] Eric Brill: Pattern-Based Disambiguation for Natural Language Processing. Proceedings of Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora. (EMNLP/VLC-2000)

[3] Christopher D. Manning and Hinrich Schütze: Foundations of Statistical Natural Language Processing, MIT Press. (2000)

[4] Van Rijsbergen: Information Retrieval. Butterworths, London. (1979)

[5] Michael Chau, Jennifer J. Xu, Hsinchun Chen: Extracting Meaningful Entities from Police Narrative Reports. Proceedings of the National Conference for Digital Government Research. Los Angeles, California. (2002)

[6] Tianhao Wu, and William M. Pottenger: An extended version of "A Semi-supervised Algorithm for Pattern Discovery in Information Extraction from Textual Data". Lehigh University Computer Science and Engineering Technical Report LU-CSE-03-001. (2003)

[7] Hopcroft, J. and J. Ullman: Introduction to Automata Theory, Languages and Computation. Addison-Wesley. (1979)