

# **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. The regular languages consist of various representations of assorted features that are useful in information extraction. An example of such a regular language is the textual representations used to express a suspect's height in a collection of police incident reports.

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. From this 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.

**Keywords:** Information Extraction, Machine Learning, Text Mining, Textual Data Mining, Supervised Learning, Regular Expression Discovery, Regular Language, Fight Against Terrorism

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

## 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.

At Lehigh University we are conducting a project with Lockheed Martin M&DS for the Pennsylvania State Police department. Our target is to develop a system that can help police to solve new cases. The first phase of the project is to extract features such as modus operandi and physical description for suspects as recorded in narrative incident reports.

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. The regular expressions discovered generate regular languages that consist of various representations of assorted features useful in information extraction.

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 when two men approached her from behind", 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 for a person's age in this particular example<sup>1</sup>. 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.

The article is organized as follows. In section 2, we summarize related work. Following this, we provide a framework for understanding our algorithm in section 3. Our approach is described in section 4, and in section 5 we detail preliminary experimental results. Finally, we present our conclusions and discuss future work in section 6 and acknowledgements in section 7.

## 2. Related Work

Although much work has been done in the field of information extraction, relatively little has focused on the automatic discovery of regular expressions. In this section, we highlight a few efforts that are related to regular expression discovery. We also discuss examples of work in the sequential pattern mining field since such approaches employ similar methods.

Stephen Soderland developed a supervised learning algorithm, WHISK [2], which uses regular expressions as patterns to extract features from semi-structured and narrative text. In each iteration of the learning process, WHISK requires that a human expert label specific features in instances and then generates rules based on these labels. WHISK uses segments such as clauses, sentences, or sentence fragments as its instances. A crucial difference between WHISK and our approach is that WHISK requires the user to identify the precise location of features for labeling

---

<sup>1</sup> Currently, only words and English part of speech tags are used in the discovery process.

while our approach requires only that instances be labeled. As noted this represents a significant reduction in the effort required to develop a training set.

Eric Brill [3] applied his transformation-based learning (TBL) framework to learn reduced regular expressions for correction of grammatical errors in text. Although Brill does not perform explicit information extraction, the correction process involves identifying grammatical errors. There are three major differences between Brill’s approach and ours. First, the reduced regular expressions generated by Brill do not include the logical “OR” operator. We have found that the “OR” operator is necessary to achieve high accuracies in information extraction. Secondly, like the aforementioned work by Soderland, Brill’s approach requires intensive feature-specific labeling to create the ground truth used in TBL. Finally, our approach does not require domain experts to create templates because it is not based on TBL.

Minos N. Garofalakis developed a system named SPIRIT [4], which finds all frequent sequential patterns in a database. SPIRIT then employs an algorithm to generate regular expressions from the sequential patterns. Similar to the templates used in Brill’s TBL-based approach, SPIRIT requires a predefined regular expression constraint as input. In essence, however, SPIRIT does not perform information extraction – it assumes an existing relational database. Our approach, on the other hand, uses the regular expressions discovered to extract information from narrative text and store it in relational form.

Helena Ahonen-Myka [5] employed a sequential pattern mining method to discover maximal sequences in text. They applied a greedy algorithm to combine bi-grams to generate all maximal sequences in which gaps could exist. There are two major differences between Ahonen-Myka’s approach and ours. One is that the maximal sequences discovered by Ahonen-Myka employ only an implicit “AND” operator between pairs of words, while our approach employs the “AND”, “OR”, and “NOT” operators. As noted previously, we have determined that these operators are necessary to achieve a high degree of accuracy in information extraction. The other difference in our approaches is that Ahonen-Myka find all maximal sequence based on frequency while our approach discovers regular expressions that are used to extract only the features of interest.

Michael Chau, Jennifer J. Xu, and Hsinchun Chen have published results of research on extracting entities from narrative police reports [9]. They employed a neural network to extract persona names, addresses, narcotic drugs, and items of personal property from these reports. Noun phrases are candidates for name entities. Although not readily apparent in [9], they evidently employ a similar approach to other researchers in that feature-specific labeling is required in training set development. Their cross-validation results vary from a low of 46.8% to a high of 85.4% for various entities. In our approach, however, we achieve significantly better results without limiting ourselves to noun phrases. In addition, we are able to extract a larger number of features that can be used for analysis in several ways, including matching on modus operandi.

In this section we summarized work in two text mining fields related to our approach. In section 3 following, we provide definitions for understanding our algorithm for discovering regular expressions used in information extraction.

### 3. 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. Following this, we define terms used in this article.

**Regular expression:** “Given a finite alphabet  $\Sigma$ , the set of regular expressions over that alphabet is defined as (Hopcroft and Ullman 1979):

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

2) 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.” [3,6]

**Reduced regular expression (RRE):** Our reduced regular expression is at first glance similar to that defined in [3]. However, there are some significant differences. Given a finite alphabet  $\Sigma$ , our reduced regular expression is defined as a set:

- 1)  $\forall a \in \Sigma$ ,  $a$  is a RRE and denotes the set  $\{a\}$ .
- 2)  $\sim a^*$  is a RRE and denotes the positive closure of the Kleene set  $\Sigma^+ a$ .
- 3)  $\wedge \in \Sigma$ ,  $\$ \in \Sigma$ , where  $\wedge$  is the start of a line, and  $\$$  is the end of a line.
- 4)  $\{ \backslash s, \backslash S, \backslash w, \backslash W \} \subset \Sigma$ , where  $\backslash s$  ( $[ \backslash t \backslash n \backslash r \backslash f ]$ ) is any white space,  $\backslash S$  ( $[ \wedge \backslash t \backslash n \backslash r \backslash f ]$ ) is any character except white space,  $\backslash w$  ( $[ 0-9a-zA-Z ]$ ) is any alphanumeric character, and  $\backslash W$  ( $[ \wedge 0-9a-zA-Z ]$ ) is any non-alphanumeric character.
- 5)  $(\backslash w)^*$  is a RRE denoting the Kleene closure of the set  $\{\backslash w\}$ .
- 6)  $(\backslash w)\{i,j\}$  is a RRE denoting that  $\backslash w$  is repeated between  $i$  and  $j$  times, where  $i=0$ , and  $j=i$ .
- 7)  $a?$  is a RRE and denotes that  $a$  is an optional part of the RRE.
- 8) 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.

**Element:** All words in the lexicon and all part of speech tags in the Penn tag corpus [7] belong to  $\Sigma$ . Each word/tag can be an element of a RRE.

**Feature:** A feature is the smallest unit of information extracted. Examples include values for the attributes *height*, *weight*, *age*, *gender*, *time*, *location*, etc.

**Segment:** A segment is (a portion of) a sentence. In our experiments we found it necessary to use periods to mark sentence boundaries and commas to mark segment boundaries [1]. The only exception is a comma that separates two numbers, as in “\$1,000”. The comma in the phrase “...in his twenties, with brown eyes...”, on the other hand, is a segment boundary. The textual string between any two segment boundaries is a segment.

**Root:** We term the first element found in an RRE the root of the RRE.

**“AND” learning process:** All learning iterations that employ the logical “AND” operator.

**“OR” learning process:** All learning iterations that employ the logical “OR” operator.

**“NOT” learning process:** All learning iterations that employ the logical “NOT” operator.

**“Optional” learning process:** All learning iterations that employ an optional operator.

**True set:** If the system is learning a RRE for a feature  $f$ , then the true set consists of all segments labeled  $f$  in training set. After each iteration in learning, the true set is updated by removing those segments that have been covered. In this sense our approach uses a covering algorithm.

**False set** If the system is learning a RRE for a feature  $f$ , then the false set consists of all segments that are not labeled  $f$  in the training set. For a given feature  $f$ , the false set does not change during learning.

$R_{\text{and}}$ : The RRE learned after completion of the “AND” learning process.

$N$ : The number of elements in  $R_{\text{and}}$ .

$C$ :  $C$  is the set of words in the lexicon combined with the part of speech tags in the Penn tag set.  $|C|$  is the number of words and tags in  $C$ .

## 4. 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 separation of the input text into segments. This step is performed automatically. Next, a domain expert labels segments with zero or more feature identifiers. Finally, we apply a greedy algorithm to discover RREs for each of the feature identifiers. We then perform 10-fold cross-validation to evaluate the performance of the model. We detail these steps in what follows.

### 4.1. Pre -Processing

Each incident report is split into segments at this stage. Each segment 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.

### 4.2. Segment Labeling

Prior to the start of the labeling stage, domain experts must identify the features that will be extracted. For instance, if the high-level goal is to extract physical descriptions of suspects, the list should include ‘Height’, ‘Weight’, ‘Eye Color’, etc. During training set development, each segment is evaluated manually and assigned the proper labels. For example, if a segment includes ‘Height’ and ‘Weight’ information, the domain expert assigns both of these labels to the segment. After labeling, each feature has its own true set and false set.

### 4.3. Part Of Speech Tagging

Part of speech tags are important in our reduced regular expression discovery algorithm. Each word in training set must be assigned its correct part of speech tag before the learning process begins. Currently, we are using Eric Brill’s part of speech tagger to tag our training sets. Brill’s tagger uses the Penn tag set [7]. We have enhanced the lexicon to include tags such as *WEEKDAY*, *CRIMETYPE*, *MALE*, *FEMALE*, etc. These tags are helpful in RRE generation of certain features. For example, the lexicon entries for “Monday” and “Friday” have the special tag *WEEKDAY* assigned. During the learning process, our algorithm discovers that the *WEEKDAY* tag is correlated with the “Day of week” feature. Table 1 list all features supported in our current system.

### 4.4. Learning Reduced Regular Expressions

The goal of our algorithm is to discover sequences of words and/or part of speech tags that have high frequency in a collection of segments, while having low frequency outside the

segments. 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 the “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 current RRE. Figure 1 depicts the entire learning process.

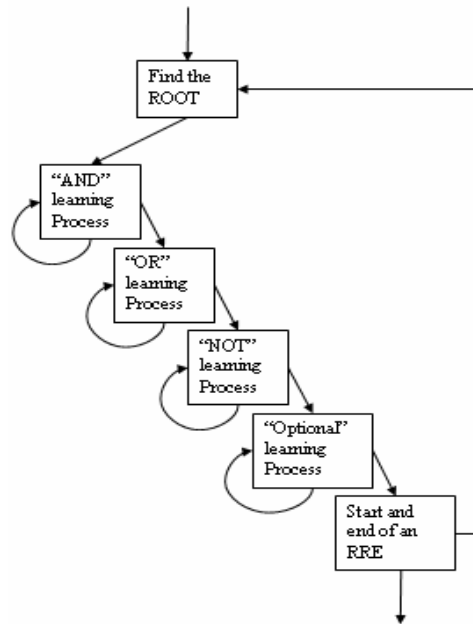


Figure 1: RREs generation process

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 use this threshold because over-fitting results if too few segments are used to discover a RRE.  $d$  is a parameter in our system, and is currently set to two. We depict the details of each step of the algorithm in what follows.

#### 4.4.1. Discovering the Root of a RRE

This step 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 F-measure (Equation 1 from [8]) 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  $b$  enables us to place greater or lesser emphasis on precision, depending on our needs.

$$F_b = \frac{(b^2 + 1) PR}{b^2 P + R}$$

Equation 1: F-measure

The word or part of speech tag with the highest score is chosen as the ‘root’ of the RRE. The algorithm discovers the word or part of speech tag that has a high frequency of occurrence in

segments with the desired feature. Meanwhile, it must also have low frequency in segments that do not contain the desired feature.

Our approach places less emphasis on precision and more on recall during the root discovery process. Naturally this results in a larger set of segments that match the root. These segments, however, are not necessarily all true positives. As a result, the “AND”, “OR”, and “NOT” learning phases all prune false positives from the set of strings that match the root RRE. The result is both high precision and high recall, or F-measure close to 100%, as shown in our results in section 5.

#### 4.4.2. “AND” Learning Process

After the root is discovered, the algorithm inserts additional words and/or part of speech tags before and after the root. 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 overall complexity of the “AND” learning process depends on both the number of candidate elements for a position in a given pass during the “AND” learning phase, and on the number of elements in  $R_{and}$ . If there are  $N$  ( $N > 0$ ) elements in a RRE, then there are  $N+1$  possible positions to insert an element. For example, if the initial “AND” learning phase learns “AB” as a RRE, then “0”, “1”, and “2” in “0A1B2” mark the three possible positions for the second “AND” learning pass to insert elements.

Equation 2 depicts the complexity of the “AND” process, where  $C_{ij}$  is the set of candidate elements for the  $j^{th}$  position in the  $i^{th}$  pass of the “AND” learning process in a single iteration of Figure 1. In the previous example,  $C_{21}$  is the set of candidates that can be used in position 1 in second pass through the “AND” learning process that produced “0A1B2”. Because there are  $N$  passes to discover a RRE of size  $N$ ,  $0 = |C_{ij}| = |C|$ . This implies that  $0 \leq C_{AND} \leq |C| \left( \frac{N(N+1)}{2} + N \right)$ . Therefore,  $O(0) = C_{AND} = O(N^2)$ . As demonstrated in our results in section 5, the training time is reasonable for the features of interest due to the fact that they are all less than 10 to 15 elements in size.

$$C_{AND} = \sum_{i=1}^N \sum_{j=0}^i |C_{ij}|$$

**Equation 2: The complexity of “AND” operation**

To improve the performance of the algorithm, we limit the candidates for each position to actual elements that occur in segments. For example, suppose we want to extend “AB” by inserting an element “\_” between “A” and “B”. The system matches “A\_B” against all segments in the current true set using only those words or part of speech tags that actually occur between “A” and “B”. This approach limits candidate elements to a small set. Moreover, the number of candidate elements becomes smaller as the RRE grows.

As noted, during the “AND” phase, our algorithm weights precision more heavily than recall. Since we employ a covering algorithm, our goal is to maximize precision during each iteration of Figure 1. Thus we employ different values of  $b$  during training to ensure that the resulting RRE is optimal.

### 4.4.3. “OR” Learning Process

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 discovered during the “AND” learning process, 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.

For example, suppose “D”, and “E” are two candidate elements to be combined with “A” in the RRE “ABC”. For the purposes of example, suppose “ABC” has an F-measure of 0.78. If “(A|D)BC” has a score of 0.8, then “(A|D)BC” will replace “ABC” as the current RRE. After that, “(A|D|E)BC” will be evaluated as an extension to “A”. The extension of “A” stops when no higher score can be found after all candidate words and part of speech tags have been evaluated. After “A” is extended, the algorithm extends “B”, and so on. This process stops after all elements in the RRE have been extended. The selection of candidate elements for the “OR” learning process is also data-driven. The time complexity of the “OR” learning process is depicted in Equation 3, where  $O_i$  is the number of candidate elements in position  $i$  (each element in  $R_{\text{and}}$  is a position for the purposes of the “OR” learning process). Since  $0 \leq |O_i| = |C|$ ,  $0 \leq C_{OR} \leq |C| \cdot N$ . Therefore,  $O(0) = C_{AND} = O(N)$

$$C_{OR} = \sum_{i=1}^N O_i$$

**Equation 3: The complexity of “OR” operation**

### 4.4.4. “Optional” Learning Process

The gap between any two adjacent elements in  $R_{\text{and}}$  between the start and the end of the RRE is a position for an optional element in a RRE. Therefore, there are  $N+1$  positions that could have optional elements. Each gap can have zero or one optional element. For each gap, the system generates a set of candidate elements using a similar method as that described in section 4.4.2. From the candidate set, the system selects one word or one part of speech tag that occurs most often in the true set. If the frequency is higher than a threshold  $\gamma$ , then the word or tag becomes an optional part of the RRE.  $\gamma$  is a parameter of our system that is currently set to half the total number of instances in the true set in a given iteration of Figure 1. The time complexity for the optional learning process is depicted in equation 4, where  $P_i$  is the number of candidate elements in position  $i$  (position  $i$  is the gap between element  $i$  and element  $i+1$ ). Since  $0 \leq |P_i| = |C|$ ,  $0 \leq C_{optional} \leq |C| \cdot (N+1)$ . Therefore,  $O(0) = C_{optional} = O(N)$

$$C_{optional} = \sum_{i=1}^{N+1} P_i$$

**Equation 4: The complexity of optional operation**

Optional elements can improve neither precision nor recall. In other words, this phase of the learning process cannot improve either training or testing F-measure scores based on segment recognition. However, we have discovered that this phase can improve performance when extracting features. For example, suppose we are interested in extracting the *height* feature. Furthermore, suppose there are only two elements, the part of speech tag “CD” and the word

“tall”, in  $R_{\text{and}}$ . This means that any number can be followed by any word or part of speech tag except “CD”, followed by the word “tall”. Meanwhile, suppose the string for a person’s *height* is “five feet six inches tall”. Unfortunately, this RRE cannot exactly match a *height* – instead it will match “six inches tall”. However, if there is an optional element “feet” between “CD” and “tall”, the RRE will achieve both optimal segment accuracy and effectively extract the feature of interest, “five feet six inches tall”.

#### 4.4.5. “NOT” Learning Process

For each element generated in the processes described in sections 4.4.2, section 4.4.3, and section 4.4.4, the system evaluates the insertion of a “NOT” operator on the element immediately following. The last element is an exception – the “NOT” operator is not applied to it because it marks the end of the RRE. For instance, if three elements found for the feature *height* are “CD”, “feet”, and “tall”, then new RREs that include  $\sim$ “CD” and  $\sim$ “feet” are generated (where  $\sim$  is the “NOT” operator). The new RREs replace RREs discovered earlier based on the F-measure score. The time complexity of applying the “NOT” operator is depicted in Equation 5. Therefore,  $C_{\text{NOT}} = O(N)$ , where  $N$  is the total number of elements in the RRE.

$$C_{\text{NOT}} = N$$

**Equation 5: The complexity of “NOT” operation**

In our implementation we used Perl. Perl, however, does not support a “NOT” operator for multi-character tokens, so we implemented the “NOT” operator as follows: we use the “NOT” operator on each single character in a word or a part of speech tag, and then use the “OR” operator to combine them. For example,  $\sim$ “feet” is expressed as “([f]|f[^e]|fe[^e]|fee[^t]|feet[^s])”.

The “NOT” operation enabled the extraction of features of interest. For example, consider a RRE that includes three elements “CD”, “feet”, and “tall”. This RRE accepts the string “weighing/NN 180/CD pounds/NNS and/CC five/CD feet/NNS six/CD inches/NNS tall/JJ”, where “NN”, “CD”, “NNS”, “CC”, and “JJ” are part of speech tags. If we use “CD(\s)\*feet(\s)\*tall” as the RRE, it will find nothing. If we use “CD.\*feet.\*tall” as the RRE, it will find “180/CD pounds/NNS and/CC five/CD feet/NNS six/CD inches/NNS tall/JJ”. If we use “CD([C]|C[^D]|CD[^s\|])\*feet([f]|f[^e]|fe[^e]|fee[^t]|feet[^s\|])\*tall([t]|t[^a]|ta[^l]|tal[^l]|tall[^s\|])\*”, the string accepted is “five/CD feet/NNS six/CD inches/NNS tall/JJ”. Obviously, the last result is the one we want for the feature *height*.

#### 4.4.6. Handling the start and the end of the RRE

If the first element found is a part of speech tag, then our algorithm ensures that the RRE also covers the word before the tag by including “(\S)\*” before the tag. For example, if the single element “CD” is discovered, then the RRE becomes ““(\S)\*{CD}””. This RRE will accept strings such as “20{CD}”.

The start symbol “^” and end symbol “\$” of a segment also proved to be useful in some cases. As a result, our algorithm tests whether the current RRE should include “^” or “\$”. We simply put “^” at the beginning of the RRE to form a new one. If it has better or equal performance compared to the previous RRE, then the RRE starting with “^” replaces the previous RRE. We deal with “\$” in a similar manner.

### 4.5. Post Processing

After each loop in Figure 1, (sections 4.41 to 4.4.6), 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 other words, given that  $R_1, R_2, \dots, R_m$  are  $m$  RREs that are discovered during learning, then the final RRE will be “ $(R_1)|(R_2)|\dots|(R_m)$ ”.

In this section we have described a greedy covering algorithm that discovers a RRE for a specific feature in narrative text. The basic idea is to find high frequency patterns in segments associated with the feature. We have applied “AND”, “OR”, and “NOT” operators to find elements of a RRE that accept sub-patterns of the feature under consideration. Optional elements as well as the start and the end of a segment are also components of the RRE. Finally, RREs for all sub-patterns are combined to form a single RRE with the “OR” operator.

## 5. 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 F-measure 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 incident reports obtained from Fairfax County, VA. The total number of incident reports we used in training was 100, which were randomly selected from a sample of over 700 reports. The 100 reports were automatically segmented into 1404 segments. Currently we support ten features in our system. Table 1 depicts the details of the 10 features as well as the number of documents and segments used in each fold of the cross-validation. Note that *Eye Color* and *Hair Color* are not well represented in our dataset due to their infrequent appearance in the Fairfax County data.

Feature	# of documents for cross-validation	# of segments for cross-validation
Age	81	141
Date	39	94
Time	82	91
Eye Color	9	10
Gender	78	336
Hair Color	8	8
Height	23	25
Race	29	34
Weekday	89	98
Weight	17	19

**Table 1: Features in the training set**

The result of the training process is one RRE for each feature. Table 2 illustrates the details of the training result. The first column depicts the features of interest in the incident reports. The second column lists all sub-patterns discovered. These are not the actual RREs but rather high-level abstractions due to the complexity of the actual RREs – they cannot easily be displayed in tabular form. In this table we list only the key elements in the RRE. “CD”, “JJ”, “IN”, “MALE”, “FEMALE”, “CDS”, “OCCURTIME”, “WEEKDAY”, “NNP”, “NNS”, “VBN”, and “VBG” are all part of speech tags. “:” and “.” are either words or part of speech tags. All others elements in these RRE abstractions are words. The final column is the average training time based on a Pentium III 1GHz processor with 320 MB of memory.

RREs generated for a given feature during cross-validation are virtually identical. This implies that our approach is stable.

Feature	Pattern expressions	Training time (seconds)
Age	CD (NN)? old ^ ([0-9a-zA-Z]){2} CD \$ in IN (MALE)? CDS (CD)? NNS (of)? IN age NN	10.7
Date	MONTH CD 2002 CD 2001 CD	7.8
Time	IN CD : : (NN)? CD . . m NN . . Fairfax NNP County (NNP)? NNP IN NNP IN OCCURTIME OCCURTIME	175.9
Eye Color	JJ eyes NNS	5.7
Gender	MALE FEMALE	11
Hair Color	JJ hair NN	5.6
Height	CD feet (CD)? NNS tall JJ	6.2
Race	described VBN (as)? IN (black)? JJ	5.9
Weekday	WEEKDAY	7.8
Weight	weighting VBG CD pounds NNS	5.8

**Table 2: RREs learned for each feature**

After completing 10-fold cross-validation, we have 10 test results for each feature. The average precision, recall and F-measure ( $\beta=1$ ) for these ten results is depicted in Table 3 and Table 4. Table 3 contains the results based on segment evaluation, and Table 4 depicts the result of testing for an exact match. We also include a column for the absolute number of true positives covered by each RRE.

*Eye Color*, *Gender* and *Weekday* have perfect test performance (100%) in part because we have modified the lexicon as noted in section 4.3. The performance of *Age*, *Date*, *Time*, *Height*, *Race*, and *Weight* are also excellent (F-measure scores =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 “2002” covers over 95% of the years in our dataset, so the algorithm discovers “2002” as a sub-pattern for “year”. After “2002” is discovered, “2001” is also discovered as a sub-pattern. After that, almost all segments in the true set have been covered. Therefore, the learning process stops. As a result, any year not occurring in the training set (e.g., “1986”) will not be recognized as a “year”. This causes the slight drop in performance for the *Date* feature.

In order to address this issue we developed a more complete training set and reevaluated our algorithm on the *Date* feature. In this 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 “year”.

Feature	Average Precision %	Average Recall %	Average F-measure %	Average # of true positives
Age	97.27	92.38	94.34	13
Date	100	94.69	97.13	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

**Table 3: 10-fold cross-validation test performance based on segment evaluation**

The performance of *Hair Color* is not very good. As noted, this is due to the lack of *Hair Color* segments in test sets (2, 4, 5, 8). However, the test performances on sets (1, 3, 6, 7, 9, 10) are perfect for *Hair Color* (F-measure score = 100%). Therefore, we conclude that the RRE discovered for the feature *Hair Color* is correct.

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 when two men approached her from behind”, 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 4.

Feature	Average Precision %	Average Recall %	Average F-measure %	Average # of true positives
Age	92.61	88	89.83	12.4
Date	100	94.69	97.13	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

**Table 4: 10-fold cross-validation test performance based on exact match**

In the exact match, *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*, *Height*, and *Weekday* all have the same performance 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.

In summary, our experimental results provide evidence that our approach can discover useful RREs that can be used to automatically extract features from incident reports. Human experts can also readily understand the decision structure of the RREs. Finally, the overall exact match performance of the RREs is quite good on our features of interest.

## 6. Conclusion

We have presented a semi-supervised learning algorithm that automatically discovers Reduced Regular Expressions (RREs) based on very 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 incident reports.

One of the tasks that lies ahead is to develop techniques similar to sentence boundary detection for use in segment boundary detection. 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.

## 7. Acknowledgements

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 for their assistance in completing this work. We are also very grateful to our families and co-workers 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] Chania, Greece .Automatic Extraction of Rules for Sentence Boundary Disambiguation In the *Proceedings of the Workshop in Machine Learning in Human Language Technology, Advance Course on Artificial Intelligence (ACAI'99)* .
- [2] S. Soderland. *Learning information extraction rules for semi-structured and free text*. Machine Learning, 34(1-3):233-272, 1999.
- [3] 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).
- [4] M. Garofalakis, R. Rastogi, and K. Shim. *SPIRIT: Sequential pattern mining with regular expression constraints*. In VLDB 1999.
- [5] Helena Ahonen-Myka. [Discovery of frequent word sequences in text](#). The ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining, Imperial College, London, 16-19 September, 2002.
- [6] Hopcroft, J. and J. Ullman (1979). Introduction to Automata Theory, Languages and Computation. Addison-Wesley.
- [7] Christopher D. Manning and Hinrich Schütze. Foundations of Statistical Natural Language Processing, *MIT Press*, 2000.
- [8] Van Rijsbergen, 1979. Information Retrieval. *Butterworths*, London

[9] Michael Chau, Jennifer J. Xu, Hsinchun Chen, ["Extracting Meaningful Entities from Police Narrative Reports,"](#) in *Proceedings of the National Conference for Digital Government Research (dgo 2002)*, Los Angeles, California, May 19-22, 2002.