Facilitating Pattern Discovery for Relation Extraction with Semantic-Signature-based Clustering

Yunyao Li  
IBM Research - Almaden  
650 Harry Road  
San Jose, CA 95120  
yunyaoli@us.ibm.com

Vivian Chu  
IBM Research - Almaden  
650 Harry Road  
San Jose, CA 95120  
vchu@cal.berkeley.edu

Huaiyu Zhu  
IBM Research - Almaden  
650 Harry Road  
San Jose, CA 95120  
huaiyu@us.ibm.com

Sebastian Blohm *  
Microsoft Corporation  
Rablstr. 26, 81669 München, Germany  
sblohm@microsoft.com

Howard Ho  
IBM Research - Almaden  
650 Harry Road  
San Jose, CA 95120  
ho@almaden.ibm.com

ABSTRACT

Hand-crafted textual patterns have been the mainstay device of practical relation extraction for decades. However, there has been little work on reducing the manual effort involved in the discovery of effective textual patterns for relation extraction. In this paper, we propose a clustering-based approach to facilitate the pattern discovery for relation extraction. Specifically, we define the notion of semantic signature to represent the most salient features of a textual fragment. We then propose a novel clustering algorithm based on semantic signature, S2C, and its enhancement S2C+. Experiments on two real-world data sets show that, when compared with $k$-means clustering, S2C and S2C+ are at least an order of magnitude faster, while generating high quality clusters that are at least comparable to the best clusters generated by $k$-means without requiring any manual tuning. Finally, a user study confirms that our clustering-based approach can indeed help users discover effective textual patterns for relation extraction with only a fraction of the manual effort required by the conventional approach.

Categories and Subject Descriptors

H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing—Linguistic processing; I.2.7 [Artificial Intelligence]: Natural language processing—text analysis

General Terms

Algorithms, Experimentation, Human Factors

Keywords

Information Extraction, Clustering, Pattern Discovery

*Work was done while visiting IBM Research - Almaden.

1. INTRODUCTION

Relation extraction refers to the task of detecting and classifying meaningful relations between two or more entities in text. For example, the text fragment “please call Alice from CompanyA Inc. at her cell phone (123) 456-7890” contains the EMPLOYEEOF relation “Alice from CompanyA Inc.” and the PERSONPHONE relation “Alice ... at her cell phone (123) 456-7890”. Relation extraction is important for many applications, ranging from information retrieval and question answering to text entailment.

Most information extraction systems rely on knowledge engineering or machine learning to generate the “task model” for relation extraction. In the knowledge engineering approach, the model is usually in the form of manually created extraction rules. In the machine learning approach, the model is learned automatically from a manually labeled training data in a supervised or semi-supervised fashion. The machine learning approach has been popular in the research community in recent years (e.g., H[12,24,32,39] [40]). At the same time, the knowledge engineering approach remains a widely adopted practical solution for relation extraction due to its transparency, customizability, and maintainability. These properties are highly valued by emerging enterprise text analytics applications (e.g. compliance and semantic search)[2][11][13][31].

In the knowledge engineering approach, a developer creating rules for a binary relationship (e.g. pairs of PERSON and PHONENUMBER) typically starts by manually examining the contexts between the pairs. She then determines whether each pair forms a meaningful relation and manually discovers textual patterns indicating the desired relation. Finally she writes rules to incorporate the patterns observed. These rules, when applied to a training corpus, usually generate a large number of results. The developer examines a (typically small) sample of the results, and determines whether each one is positive or negative. She then discovers potential improvements to the rules by examining the relationship between the results, the original context strings, and the applicable patterns. She iterates through the process until she is satisfied with the precision and recall of the relation extraction annotator. The following example illustrates one step in this process.

EXAMPLE 1. PERSONPHONE. An obvious pattern for identifying the PERSONPHONE relation is “PERSON followed by ‘at’ followed by a PHONENUMBER within the same sentence”, represented as $P_1 = \langle (PERSON) \cdot \at \cdot (PHONENUMBER) \rangle$. While $P_1$
can correctly identify valid PERSONPHONE relation such as “Alice at (123) 456-7890” and “Bob can be reached at (111) 222-3333”, it not only produces incorrect matches such as “Jane’s cell: (222) 333-4444”. In order to account for precision and recall, a rule developer needs to refine \( P_1 \) (e.g. \( P_2 = "\text{(PERSON)}\text{(can be reached) at } \ast \text{ (PHONENUMBER)}\)”) as well as identifying additional patterns (e.g. \( P_3 = "\text{(PERSON)}\text{’s (cell|office|home)}\) ? : (PHONENUMBER)”).

As can be seen, this is an extremely tedious and error-prone manual process that can take days or even weeks of work to develop rules for a single relation. The focus of this work is to reduce the human effort involved in writing rules for relation extraction. In this paper, we propose a novel formulation of clustering algorithm called Semantic-Signature Clustering (short as S2C) to facilitate the discovery of patterns for relation extraction. In the rest of the section, we first summarize our contributions and then discuss in detail the related work.

1.1 Contributions

In a key departure from prior formulations, the clustering problem presented in this work not only clusters similar strings, but also generates meaningful patterns. These patterns are similar to the manually generated ones used in relation extraction rules. Our specific contributions are:

- A novel clustering problem consisting of clustering strings of relation candidates and generating clusterings that share the same pattern for relation extraction.
- Formulation of this clustering task as a three-step algorithm S2C, where each of these steps can be optimized individually with well-known techniques:
  - (Potentially offline) sequence mining & rule generation
  - Efficient rule application to generate semantic signatures
  - Efficient database-like grouping based on exact match
- Extensive experimental results over real-world data sets that showcase the effectiveness of S2C. The results include (1) improved precision and recall of relation extraction rules written with the assistance of S2C + over product-quality relation extraction rules generated in a purely manual fashion; (2) S2C + runs an order of magnitude faster than conventional \( k \)-means clustering for real-world relation extraction tasks.

Conventional text clustering algorithms (e.g. \( k \)-means [27]) are based on computing the similarity of context strings among each other or between each context string and a cluster representative. Context strings are mapped to a space (e.g. converted to a numeric vector) in which such a distance can be computed. For large scale relation extraction, both the numbers of context strings (depending on the size of the corpus) and potential clusters representatives (one needed for each way to express a relation) grow rapidly to large sizes and causes the number of comparisons to grow rapidly. As we will show later, our clustering method saves these comparisons by only requiring a rule-based local transformation on the context strings. Furthermore, unlike conventional clustering algorithms, our clustering method requires no extensive manual tuning on the parameters. To further motivate our approach, we now discuss prior work with a focus on related work in the area of clustering for relation extraction and the limitations of these techniques.

1.2 Related Work

Textual patterns play an important role in relation extraction [6, 21]. As discussed earlier, learning of patterns for relation extraction is typically a supervised process in learning-based systems or a tedious manual process in knowledge-engineering-based systems. Several semi-supervised systems were developed to enable relation extraction using a large unlabeled corpus and a small set of seeds [1].

A number of recent works focus on unsupervised relation identification. [20, 7] discover relations of co-occurring named entities by clustering based on the words that appear between each pair. [13] groups textual occurrences based on the so-called hook words and then uses statistical correlation between the related entities contained in the textual examples to generate meaningful groups. Another clustering based method was proposed by [33] where the goal is to identify relations by clustering pairs of entities. Classical sequence patterns are mined as part of this approach to generate features for clustering. The SNE system [25] clusters the subject-verb-object triples generated by an Open Information Extraction system into semantically coherent relations by probabilistically modeling the triples in second-order Markov logic and applying a co-clustering algorithm.

While unsupervised relation identification can reduce the labor cost of semi-supervised systems by automatically generating seeds [34], little work has been done for the building of relation extraction patterns used by the knowledge-engineering-based systems. Our work seeks to fill this gap. It takes advantage of the unsupervised nature of clustering method similar to unsupervised relation identification, but with the goal of facilitating pattern discovery for relation extraction for knowledge-engineering-based systems. Our clustering method is configurable by the user using intuitive features, as the default clustering may or may not produce the granularity desirable to a user.

Our frequency-based pattern generation technique was motivated by the vGram approach [26] which performs fuzzy string matching by means of an automatically mined set of variable length subsequence. Our approach is based on a modification of the Apriori algorithm [2]. A similar approach is applied to web usage log mining in [29]. Their technique is based on task-specific pruning of the pattern candidate space. We extend their technique to mine interesting pairs of sequences. The underlying generalization of Apriori is described in [35]. An application of a more basic sequence mining technique to sentiment detection in texts can be found in [23].

2. OVERVIEW

In this work, we are primarily concerned with the task of clustering the contexts extracted from the vicinity of relation candidates to assist the discovery of patterns used for relation extraction. Once these clusters are formed the patterns may be generated either manually by an annotator developer or automatically by existing techniques such as [6, 16]. This second phase will not be discussed further here, but we do present user study results in the experimental section to demonstrate the effectiveness of the generated clusters in assisting annotator developers.

For pedagogical convenience, in the rest of the paper we restrict our attention to only consider binary relations and only context strings between the two entities in the relation. It will be obvious that the techniques presented here are applicable to more general situations. For example, the context string could include the complete sentence(s) in which the two entities appear, which may extend to the left and right of both entities.

Under the above restrictions, the clustering problem for a target
Figure 3 shows the results of the transformations in each step.

Section 3. SEMANTIC SIGNATURE GENERATION

These stages are described in Section 3 and 4 respectively.

The second stage utilizes textual similarities of the context strings.

The first stage utilizes the statistical correlations of the context strings.

Figure 2 provides three example inputs. We use data in

3. SEMANTIC SIGNATURE GENERATION

The semantic signature generation stage consists of several steps, as illustrated in Figure 4. We describe details of each step in this section. Figure 2 provides three example inputs. We use data in the first row to construct a running example for this entire section. Figure 3 shows the results of the transformations in each step.

Step 1: Generating Frequent Sequences.

The goal of this step is to map the context strings to a reduced set of sequences that are intended to capture the most salient features of the context strings. This mapping is done in the following process with three parameters: a tokenization procedure \( T \), an integer \( l \) for maximum length of sequence, and an integer \( f \) for minimum frequency of the sequence. The steps are: (1) Tokenizing each context string \( s_c \) by \( T \) into a token sequence \( s \). (2) Collecting all subsequences of \( s \) with length no more than \( l \) generated from all \( s_c \in S_C \). (3) Retaining all those with at least \( f \) occurrences in the corpus. The result is a frequent sequence set \( S_{C,T,f} \). In actual implementation, a single pass through the data is sufficient.

Step 2: Computing Correlation.

Once each context string is reduced to a set of sequences, we can collect statistics to compute the correlation between sequences. By discovering the correlation between sequences, similar sequences within the same set can be removed. This results in a semantic signature that represents the key elements of the context string. Different measure of correlation can be used in this step. In this paper, we choose uncertainty coefficient \( U(x,y) \) to be the measure of correlation between two sequences.

For each ordered pair of frequent sequences \( x \) and \( y \), the proportion of information in \( x \) that is shared with \( y \) can be measured by the uncertainty coefficient

\[
U(x|y) = I(x,y)/H(x),
\]

where \( I(x,y) \) is the mutual information between \( x \) and \( y \), and \( H(x) \) is the entropy of \( x \). If \( U(x|y) \) is close to unity, dropping \( x \) in the presence of \( y \) will not remove a significant portion of available information. In practice, we can approximately compute \( I(x,y) = p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \), and \( H(x) = -p(x) \log p(x) \), where \( p(x) \) and \( p(x,y) \) are the empirical probabilities of occurrences and co-occurrence. The definitions of \( I(x,y) \) and \( H(x) \) involve summing over the probabilities of occurrence and non-occurrence. However, since our interest here is only in the effect of occurrences of the sequences, and since the probability of non-occurrence is much larger than that of occurrence, terms for non-occurrences can be ignored.

Step 3: Generating Drop Rules.

For each pair of \( x,y \in S_{C,T,f} \), if \( U(x|y) \) is larger than a predefined threshold \( t \) we generate a “drop rule” \( \text{DROP}(x|y) \). They are stored in decreasing order of corresponding \( U(x|y) \). Figure 4 provides an example of some uncertainty coefficients and the corresponding drop rules generated. These will be used later in our running example.

Step 4: Removal of Subsumed Sequences.

To further remove noise from the semantic signature, we remove those sequences \( x \) in the presence of other sequences \( y \) if the orig-
Step 5: Applying the Drop Rules.

The drop rules are applied to remove non-informative sequences. As described earlier, these rules were generated to indicate that one sequence is not informative in the presence of another sequence. For instance, the current set \{n\; manager \; \text{tel}\} results in \{n; manager \; \text{tel}\} when the rule \text{DROP}(n; manager \; \text{tel}) is applied.

The rules are applied in their stored order, which is determined by the correlation measure associated with each rule. Applying the drop rules in this order guarantees that the sequence with less useful information is dropped. For instance, if both \text{DROP}(x|y) and \text{DROP}(y|x) exist where \text{DROP}(x|y) has a higher measure of uncertainty, then if the sequence set includes \{x, y\}, x is removed from the set and \text{DROP}(y|x) has no effect. An example of this appears in the following step.

One may wonder why we chose to apply the drop rules after removing subsumed sequences, instead of before. When generating statistics, the results are generated based on the correlation between all pairs of sequences without considering the subsumed relationship. However, when applying the rules, we take a conservative approach and require a subsequence to be as important by itself as it is with other subsequences. If rules are applied before, essentially a higher weight is given to each subsequence and important sequences can be lost. Take, for example, the sequence set \{a, ab, c\} where \text{ab} represent a sequence comprised of the subsequences a and b. If \text{DROP}(c|a) is applied before subsumed, the resulting signature is \{ab\} and c is lost. However, if subsumed is applied first, which removes a due to the presence of \text{ab}, the resulting signature is \{ab, c\}, preserving c. Only an explicit rule like \text{DROP}(c|ab) would remove c.

Step 6: Applying the drop rules to split sequences.

For certain sequences, a second pass with the drop rules is required, after all larger sequences are split into subsequences of length 1. This captures cases where a sequence of length 1 is capable of describing the entire context string, but has been stored together with a second sequence of less importance.

For the previous sequence set \{n; manager \; \text{tel}\}, the only rule that would apply is \text{DROP}(n; manager \; \text{tel})). However, by splitting the larger sequences, the rules in Figure 4 can now remove sequences that do not contribute meaningful information to the final semantic signature. Applying the rules, in stored order, results in a final sequence string of \{n; \text{tel}\}. Notice that when the rules are applied in stored order, \{manager\} was dropped as opposed to \{\text{tel}\}.

Step 7: Clustering based on semantic signature.

We call the resulting sequence of tokens representing each context string the semantic signature of the original string.

Once we obtain the semantic signature of each context string for all the relation candidates, the clustering of context strings \text{S2C} is straightforward — all the relation candidates associated with the same semantic signatures form one cluster. Candidates with different semantic signatures are in different clusters. In addition, any cluster with size smaller than a predefined threshold \(t\) is reassigned to a special cluster \(O_t\). We refer to this clustering method as \text{S2C}.

Figure 3 illustrates sample entries from two clusters of relation candidates generated by \text{S2C}. As can be seen, the entries in each cluster are indeed semantically similar to each other, as represented by their semantic signature, \{number\}, \{at\}, respectively.

Figure 4: Example drop rule generation

\begin{tabular}{l|l|l|l}
\hline
x & y & U(x|y) & U(y|x) \\
\hline
\{manager\} & \{tel\} & 0.962 & 0.802 \\
{manager \; \text{\&\; tel}\} & \{'n\} & 0.762 & 0.553 \\
\hline
\end{tabular}

(a) Example uncertainty coefficients

\begin{tabular}{l|l}
\hline
U(x, y) & Drop Rules \\
\hline
0.962 & \text{DROP}(\{manager; \text{tel}\}) \\
0.802 & \text{DROP}(\{\text{tel}; \text{manager}\}) \\
0.762 & \text{DROP}(\{'n'; \text{manager \; \&\; tel}\}) \\
\hline
\end{tabular}

(b) Example drop rules generated with threshold \(t = 0.75\) and stored order of \(U(x|y)\)

Figure 5: Sample entries of two clusters generated by \text{S2C}

\begin{tabular}{l|l|l}
\hline
Size & Sample Original Context Text & Semantic Signature \\
\hline
19 & \{'s\ number\ is\ \{number\} \&\ and\ her\ phone\ number\ is\ \{number\} \&\ number:\ \{number\} \&\ 's\ cell\ number\ \{number\} \&\ at\ \{at\} \&\ with\ at\ questions\ at\ \{at\} \&\ tomorrow\ at\ \{at\} \\
2 & \{'n\; \text{Domestic\ Number}\; \{domestic\ number\} \\
1 & \{He\ said\ the\ number\ is\ \{number\} \\
\hline
\end{tabular}

Figure 6: Example of three similar clusters generated by \text{S2C}, which are merged into one cluster by \text{S2C+}

4. CLUSTERING AND APPLICATION

In this section, we discuss the issues of applying the \text{S2C} results to assist in annotator development.

4.1 Merging of small groups based on string similarities

One remaining problem with the \text{S2C} algorithm is that the “Orphan” cluster \(O_t\) can be a non-trivial size. Recall that \(O_t\) is a conglomerate of all clusters with size smaller than \(t\). It is clear that the content of \(O_t\) is very diverse and difficult to discover meaningful patterns from. Consequently, when \(O_t\) is large the resulting relational annotator developed with the help of this clustering may have a poorer recall. In order to reduce the size of \(O_t\), we utilize the observation that the semantic signatures of many small clusters are often similar to those of larger ones. For instance, Figure 6 lists three different clusters created by \text{S2C}, with different but similar semantic signatures. If we merge some of the small clusters with larger ones, the total number of clusters stays the same, but the
size of $O_t$ will decrease. As long as the semantic signatures of the merged clusters are similar, the clusters can still assist the pattern discovery for relation extraction.

Merging small clusters into larger clusters, rather than simply discarding them, improves the results in two ways. First, after merging we still retain the set of original semantic signatures. This provides the user with very useful hints on the possible variations of semantic signatures. Second, the user can also drill down to all these clusters to see actual examples, which is a very crucial requirement in example-based rule development.

We measure the distance between two clusters by the distance between their respective semantic signatures, which are sets of token sequences. There are several ways to measure the (dis)similarity between them. For example, we can use the Jaccard distance on these two sets. Given such a distance $d$, we proceed in the following steps: (1) For all the small clusters $C$ that went into $O_t$, we look for those with size smaller than $O_t < t$. (2) If there exists a cluster $C_j$ with size larger than $t_l$ such that $d(C; C_j) > s$, remove $C$ from $O_t$ to $C_j$. When there are multiple $C_j$ satisfying the condition, we pick the largest among them. Other policies might be advantageous in certain aspects and further research is needed to quality them.

For example, when given $t_a = 3$, $t_l = 9$ and $d$ the Jaccard Distance, the three clusters in Figure 6 are merged into one.

4.2 Complexity of the $S2C$ and $S2C+$ algorithms

The computation in $S2C$ involves several pass through the data points (context strings). In step 1 (generating frequent sequence), the set has $O(nml)$ cost, where $n$ is the number of data points, $m$ is the average number of tokens generated from each context string, and $l$ is the maximum length subsequences allowed. Steps 2 and 3 (computing uncertainty coefficients and drop rules) have $O(ns^2)$ cost, where $s < ml$ is the average number of frequent subsequences for each context string. Seps 4 and 5 (removing subsumed subsequences and of applying drop rules) each also has $O(ns^2)$ cost. Step 6 (applying drop rules on split strings) has an $O(nms)$ cost. The final clustering step has $O(nC)$ cost. Therefore the total cost is $O(ns^2) + O(nms)$. Note that both $m$ and $s$ are only dependent on the distribution of the context strings and the parameters used in the algorithm. They do not depend on the size of the data set. Therefore the cost of this algorithm is essentially linear in the size of its input data.

The computation in $S2C+$ involves sweeping through the small clusters and computing their distance to each of the large clusters. The cost is $O(k_1k_2d)$, where $k_1$ and $k_2$ are the numbers of large and small clusters, and $d$ is the cost of calculating a distance between two semantic signatures. If we choose the cutoff threshold $t_l$ by limiting the number of large clusters to at most $L$, then the cost is bounded by $O(nLd)$. Since $L$ and $d$ are independent of the data size, the cost of $S2C+$ is also linear in the size of input data.

In summary the computational complexity of the $S2C$ and $S2C+$ algorithms are both linear in the input data size.

In contrast, the $k$-means algorithm is known to have a complexity $O(Iknt)$, where $I$ is the number of iterations, and $t$ is the time needed to calculate the distance between two points.

4.3 User interface for utilizing the clustering results

We designed a novel GUI to facilitate the exploration of the clusters generated by $S2C$ or $S2C+$ for pattern discovery [3]. See the screenshot in Figure 7. The basic idea is to allow a developer to quickly identify a cluster of interest based on its size and semantic signature and then have the ability to drill down and explore the actual relation candidate contained by the cluster. In addition, through the GUI, the user should be able to exploit any labeling information available.

One of the most important design decision of the $S2C$ system is that it can be a useful tool in the development cycle of annotator development. This imposes a constraint on system response time. As will be shown in the experimental section, $S2C$ is at least an order of magnitude faster than $k$-means. In addition, as many of the steps in the $S2C$ process are reusable, the actual response time of $S2C$ embedded in an annotator development environment is much shorter than the end-to-end running time. In many cases, the computation of frequent sequences and drop rules need to be computed only once. The user adjustable parameters are the thresholds for applying drop rules, for the orphan group, and for merging the small groups. Re-computation when these parameters change are almost instantaneous and are easily controlled in the GUI.

5. EXPERIMENTS

In this section, we present an empirical study of the $S2C$ and $S2C+$ algorithms over two real-life data sets. The main goal of the study is answer the following questions: (1) Do the algorithms generate high quality clusters for relation extraction? (2) Are the algorithms effective in reducing manual effort in discovering textual patterns for high-quality relation extraction? (3) How do the algorithms compare with existing clustering algorithms, in both the quality of the clusters created as well as runtime performance?

5.1 Experimental Setup

5.1.1 Data Sets

For the experiments, we used two different real-life text corpora:

- **Bio**: A collection of 2,068 biographies from 456 SEC Form DEF 14A filings [1].
- **Email**: A collection of 37,939 email messages from the publicly available Enron collection [28].

The former contains formal clean text, while the latter contains mostly informal noisy messages.

1The data set is now publicly available at URL.
5.1.2 Relation Extraction Tasks

For each data set, we chose a real-life relation extraction task that has been previously implemented for commercial products via a knowledge-based approach. Specifically, for the Bio data set, we chose the task of identifying POSITIONINORG and for the Email data set, we chose the task of identifying PERSONPHONE. The goal of our study is to compare the patterns produced manually, with ones crafted with the assistance of S2C and S2C+ in terms of the extraction quality of patterns as well as the amount of manual effort required to derive these patterns.

5.1.3 Gold Standard

We obtained the labeled data via the use of Amazon Mechanical Turk (MTurk), which has been shown to produce high quality annotations for a variety of natural language processing tasks [15]. Following the practice by [17], we first used existing state-of-the-art NER annotators [10] to identify Person and PhoneNumber over Email, Position and Organization over Bio. We then obtained noisy potential relation candidates by extracting (Person, PhoneNumber) and (Position, Organization) pairs from the data sets, with each pair of entities within 40 tokens of each other. We submitted each candidate along with its context as a Human Intelligence Task (HIT) to MTurk and asked the workers to select from three annotation options: the candidate (1) represents the relation, (2) does not represent the relation, or (3) not applicable.

We initially launched a pilot study with 100 HITs (i.e., 100 relation candidates). Upon the completion of the pilot and verification of the quality of the result, we then deployed the full study for all the relation candidates. Each HIT was assigned to five unique workers for a cost of $0.02 each. A total of 5171 HITs were submitted each candidate along with its context as a Human Intelligence Task (HIT) to MTurk and asked the workers to select from three annotation options: the candidate (1) represents the relation, (2) does not represent the relation, or (3) not applicable.

5.1.4 Comparison Study

In our experiments we evaluated both S2C and its enhancement S2C+ with $t_s = 3$ and $t_l = 10$. We also compared them to k-means [27], a popular generic clustering algorithm, using the implementation from WEKA [19]. Note that the results of k-means is sensitive to $k$, the number of clusters to be generated. Since we do not know about the optimal number of $k$, we set $k$ of k-means to be the number of clusters found automatically by S2C and S2C+, denoted as $k$-$\text{means}_1$ and $k$-$\text{means}_2$ respectively. For all the clustering algorithms, we posed the threshold for orphan cluster $f = 5$ and move all the clusters with size smaller than $t$ into a special orphan cluster.

5.1.5 Experimental Platform

We ran all our experiments on a Windows-based PC with Intel Core 2 Duo processor with a single core speed of 2.20GHz and total RAM of 3.0 GB.

5.2 Experiment 1: Quality of Clusters

We first examined the quality of the clusters generated by S2C and S2C+ in two aspects: (1) the total number of clusters that the algorithm generates, which determines the number of textual patterns required to develop the relation extraction annotator; (2) the overall quality of the clusters.

5.2.1 Plausible Reliability

Intuitively, the quality of a cluster correlates with how reliable a (positive or negative) textual pattern can be generated based on the cluster. Therefore, given the precision of a cluster $C$, $\text{Precision}(C) = \frac{|C \cap G|}{|C|}$, where $G$ is the gold standard, the closer $\text{Precision}(C)$ to 1 or 0, the higher the quality of $C$. Following this intuition, we propose a new metric called Plausible Reliability (short as PR) to measure the quality of $C$, based on how close $P(C)$ is to 1 or 0, defined as follows:

$$\text{PR}(C) = \max(1 - \text{Precision}(C), \text{Precision}(C))$$  (2)

Hence, the value of $\text{PR}(C)$ for a cluster $C$ is between 0.5 and 1. The higher the value of $\text{PR}(C)$ is, the better the quality of $C$.

To measure the overall quality of a set of clusters $C = C_0, \ldots, C_n$, we can use their average $\text{PR}$, defined as follows:

$$\text{Average PR}(C) = \frac{\sum_{i=0}^{n} |C_i| \times \text{PR}(C_i)}{\sum_{i=0}^{n} |C_i|}$$  (3)

5.2.2 Results over Bio

We first analyze the results obtained over Bio for the four different techniques. Figure 8(a) shows the histogram of clusters of
different plausible reliability. For all four techniques, the number of clusters is not large, ranging from 70 for S2C and S2C+ and 89 for \( k \)-means\(_1\). More importantly, the majority of the clusters (ranging from 68.5% for S2C+ to 80.0% for \( k \)-means\(_1\)) are high quality clusters with PR higher than 0.9. Furthermore, the vast majority of the clusters (around 90% across the board) are at least good clusters, i.e. clusters with PR higher than 0.8.

Another important aspect of the quality of the clusters is their coverage, especially of the high quality ones. Figure \[\text{9}b\] depicts the distribution of data by plausible reliability. As we can see, the high quality clusters generated by all techniques cover the majority of the data (ranging from 72% to 76%). Furthermore, good quality clusters, i.e. clusters with PR higher than 0.8, cover the major of the remaining data. Recall that in the experiment, we provide \( k \)-means the cluster sizes found using S2C and S2C+ for \( k \)-means\(_1\) and \( k \)-means\(_2\) correspondingly. We found that on this data set, \( k \)-means\(_1\) and \( k \)-means\(_2\) produced higher number of high quality clusters than S2C and S2C+ respectively. We also found that the clusters produced by \( k \)-means\(_1\) and \( k \)-means\(_2\) have higher coverage than those produced by S2C and S2C+ respectively, but the advantage is very small.

We can see from Figure \[\text{9}b\] that as expected, S2C+ has produced fewer high quality clusters than S2C, as more noise has been introduced into the clusters when merging small clusters with larger ones in S2C+. Meanwhile, we also observe that S2C+ has effectively reduced the size of the orphan cluster and evidently improved the coverage of the clusters over S2C.

Since the result of \( k \)-means is sensitive to the given value of \( k \), in order to verify that the automatically found cluster sizes were reasonable parameters for \( k \)-means, we evaluated the performance of \( k \)-means on the Bio corpus with varying \( k \)-parameters. The results are summarized in Figure \[\text{9}a\]. As can be seen, the average PR of (weighted by the number of entries contained by each cluster) the clusters produced by \( k \)-means increases gradually with the increase of \( k \), while the coverage of the clusters drops steadily as well. The graph also indicates that the size automatically discovered by S2C+ provides a good trade off between the coverage of the clusters and the overall quality of the clusters.

5.2.3 Results over Email

The performance of the algorithm over the Email set are analyzed similarly and the results are shown in Figure \[\text{10}\]. Similar to their performance over Bio, all algorithms generate relatively small numbers of clusters, with most being high quality clusters that together cover most data. Compared to the results over Bio, the number of high quality clusters generated by all techniques increases. However, their coverage drops considerably across the board. The drop is especially significant for \( k \)-means\(_1\) and \( k \)-means\(_2\), both from a coverage of lower 70% over Bio to that of lower 40% over Email. When taking good quality clusters into consideration, the coverage of \( k \)-means\(_1\) and \( k \)-means\(_2\) over Email remains lower than that over Bio. However, the coverage of S2C and S2C+ actual improves from around 70% on Email to nearly 80% on Bio.

S2C and S2C+ consistently outperform \( k \)-means\(_1\) and \( k \)-means\(_2\) respectively both in terms of the good quality clusters generated and in terms of the coverage of the clusters. The advantage is especially pronounced when we consider the coverage for clusters with PR higher than 0.8, where the coverage of good quality clusters of \( k \)-means\(_1\) and \( k \)-means\(_2\) is drastically lower than ones generated by S2C and S2C+. The decay of the performance of \( k \)-means\(_1\) and \( k \)-means\(_2\) on Email is not surprising — Email is much more noisy than Bio, and as a result, clustering blindly based on all the string content leads to less reliable clusters.

Once again, S2C+ does introduce noise to the clusters, with the coverage of high quality clusters dropping slightly from 56.8% to 54.5%. But at the same time, it effectively reduces the number of clusters that the developer needs to evaluate.
5.3 Experiment 2: Effectiveness of S2C

Results of Experiment 1 indicate that S2C and S2C+ are able to automatically generate high quality clusters with good coverage without requiring manual tuning. Intuitively, the clusters generated, such as shown in Figure 11, should be able to help a rule developer generate high quality extraction patterns with reduced manual cost. To evaluate whether this main goal of our work is achieved, we conducted the following user study on the effectiveness of S2C algorithms.

In this experiment, each data set is randomly partitioned into 80% training and 20% test data. Two rule developers were asked to independently write the PERSONPHONE annotator for Email and POSITIONINORG annotator for Bio based on the clusters generated by S2C+ over the training data. To quantify the amount of manual effort involved in the annotator development, each developer was given 2 hours for each task. We then compared the extraction quality of the annotators developed with the help of S2C+ with the corresponding product-quality annotators developed previously using conventional purely manual approach over the entire data set.

The results are summarized in Table 1 and 2. As can be seen, even though the developer using the purely manual approach was given much longer time (2 to 3 weeks vs. 2 hours) than the developers using the S2C+ assisted approach, the extraction quality of the annotators created in the conventional approach (Manualbio) is significantly lower than that of the annotators developed with the assistance of S2C+ (S2Cbio and S2Cemail) in recall and F-measure. In addition, the precision of the annotators developed by S2C+ assisted approach is at least comparable to, if not better than, the precision of the conventional manual approach. Specifically, the precision of Manualbio is slightly better than that of S2Cbio, while the precision of Manualemail is considerably lower than that of the annotator developed with assist of S2C (S2Cemail). The results confirm that our S2C algorithms are indeed effective in reducing the manual effort required for developing high quality relation annotators.

In addition, Figures 12 and 13 show the progression of improvement each time the developer added a new pattern. As one can see, the recall on both data sets steadily increases while precision is held consistently high. Furthermore, Table 3 lists the quality results of the annotators over both training and test data on Bio. The quality of the annotator over the test data set are comparable to that over the training data set, implying that no over fitting has occurred.

5.4 Experiment 3: Running time

As discussed earlier, one motivation of our work on developing the new S2C-based clustering algorithms is to be able to produce clusters with high efficiency. We therefore also measured the runtime performance of k-means and S2C in our comparison study.

We found that for both Bio and Email, it took k-means at least one order of magnitude longer to return results than S2C or S2C+.

### Table 1: Extraction quality over Bio

<table>
<thead>
<tr>
<th>Annotator</th>
<th>Precision(%)</th>
<th>Recall(%)</th>
<th>F0.5(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manualbio</td>
<td>98.37</td>
<td>53.68</td>
<td>84.33</td>
</tr>
<tr>
<td>S2Cbio Expert 1</td>
<td>97.83</td>
<td>90.42</td>
<td>96.25</td>
</tr>
<tr>
<td>S2Cbio Expert 2</td>
<td>98.20</td>
<td>84.86</td>
<td>95.21</td>
</tr>
</tbody>
</table>

### Table 2: Extraction quality over Email

<table>
<thead>
<tr>
<th>Annotator</th>
<th>Precision(%)</th>
<th>Recall(%)</th>
<th>F0.5(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manualemail</td>
<td>90.44</td>
<td>63.36</td>
<td>83.30</td>
</tr>
<tr>
<td>S2Cemail Expert 1</td>
<td>92.69</td>
<td>68.96</td>
<td>86.72</td>
</tr>
<tr>
<td>S2Cemail Expert 2</td>
<td>91.70</td>
<td>89.26</td>
<td>91.20</td>
</tr>
</tbody>
</table>

### Table 3: Statistics for training set vs test set on Bio

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Precision(%)</th>
<th>Recall(%)</th>
<th>F0.5(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expert 1 Training</td>
<td>97.36</td>
<td>88.97</td>
<td>95.56</td>
</tr>
<tr>
<td>Expert 1 Test</td>
<td>97.83</td>
<td>90.42</td>
<td>96.25</td>
</tr>
<tr>
<td>Expert 2 Training</td>
<td>97.39</td>
<td>86.03</td>
<td>94.88</td>
</tr>
<tr>
<td>Expert 2 Test</td>
<td>98.20</td>
<td>84.86</td>
<td>95.21</td>
</tr>
</tbody>
</table>

For instance, over the Email corpus, it took k-means 50 minutes to generate all the clusters compared to the 5 minutes needed for S2C. Such long delay in generating results is problematic, since the clustering algorithms are embedded in an interactive annotator development tool where developers may desire to make changes to the configuration parameters to generate optimal clusters. Furthermore, S2C can often reuse the statistics previously computed to further speed up the cluster generation process based on various optimization techniques similar to database optimization. For instance, if the developer only makes the threshold for uncertainty coefficient stricter, the algorithm can determine that the only steps from drop rule generation need to be recomputed. Finally, k-means requires the user to run the clustering multiple times to find the optimal k-parameter and can potentially cause further delay in cluster generation.

We also observed that k-means tends to consume much more heap-based memory that S2C. When running our more complicated data set Email, often times k-means would not return a result because it would eventually consume over 1.5 GB of memory. In contrast, S2C never consumes over 1 GB of memory throughout our experiments. One reason is that our intermediate steps are saved to a database to conserve heap memory.

Given the fact that most of our annotator development are done on a laptop-based environment similar to the one used in our experiments, we can see that S2C is more suitable to assist the pattern
discovery task for relation extraction for both its faster response time and lower memory consumption.

5.5 Discussions of results

In Experiment 2, the annotators developed with the assist of S2C+ significantly outperform those developed in the conventional approach, while requiring a fraction of the development effort. The high quality clusters and their informative semantic signatures provided effectively assistance in the pattern discovery process, reducing the development time for a high-quality relation extraction annotator from weeks using the conventional purely manual approach to merely hours using our new approach. Two additional issues were identified that may further speed up the development process, as will be discussed briefly in the following.

One issue concerns the usability of the clustering-based pattern discovery tool. Both expert annotator developers found that our GUI was effective in helping them understand and explore the clusters. However, they reported difficulty in keeping track of clusters that are already covered by their existing patterns, which resulted in wasted time developing additional patterns. One way to address this issue is to integrate the tool into the annotator development environment, so that the coverage of the patterns crafted by the developer is reflected automatically in the visualization of the clusters.

The other issue is that while the semantic signature of a cluster provides valuable starting point for the generation of the patterns for the cluster, the actual pattern generation is still a manual process. A promising future direction is to generate those patterns in an automatic or semi-automatic fashion. As we have discussed in the relation work, there are a number of existing works on pattern induction based on a small number of seeds [4, 6, 15, 18, 30, 37]. It would be interesting to integrate those techniques into the context of a more interactive pattern discovery tool.

6. SUMMARY AND FUTURE WORK

We proposed a clustering-based approach to assist the pattern discovery process, and presented two novel algorithms, S2C and its enhancement S2C+, to automatically cluster relation candidates based on their semantic signatures. Our extensive experimental study has confirmed the practical impact of this approach — with the assist of the clusters, a rule developer can craft product-quality annotators for relation extraction with significantly lower manual effort than that required by the conventional manual approach. Furthermore, S2C and S2C+ not only run an order of magnitude faster with less memory required than k-means, they also generate clusters of a comparable or better quality.

As described earlier, we have fully implemented our semantic-signature-based clustering algorithms with a user-friendly GUI. In fact, this pattern discovery tool is being transferred into a commercial product. For future work, we plan to focus on the two issues
discussed in Section5 to further improve the usability of this tool and to explore the feasibility of automating the pattern generation process based on the clusters.

7. REFERENCES


