Semi-Automatic Generation of Linear Event Extraction Patterns for Free Texts

© Daria Dzendzik
HP Labs
daria.dzendzik@hp.com

Sergey Serebryakov
HP Labs
sergey.serebryakov@hp.com

Abstract

In this paper we describe a semi-automatic approach to generating event extraction patterns for free texts. The algorithm is composed of four steps: we automatically extract possible events from a corpus of free documents, cluster them using dependency-based parse tree paths, validate random samples from each cluster and generate linear patterns using positive event clusters. We compare our algorithm with the system that uses manually created patterns.

1 Introduction

The event extraction (EE) task is formulated as "to automatically identify the events in free texts and derive detailed information about them, ideally identifying who did what to whom, when, with what methods, where and why". Events involve entities and relations between them and imply a change of state. For instance, the sentence "Palm was acquired by Hewlett-Packard for $1.2 billion two days ago" mentions an event of the type *Mergers & Acquisitions* (*M&A*) with four arguments - acquirer (Hewlett-Packard), acquiree (Palm), monetary expression ($1.2 billion) and temporal expression (two days ago). There is also an event indicator (was acquired). An event indicator is a word or a sequence of words that clearly signal about the possible presence of an event.

EE has been the subject of active research for more than twenty years since the series of MUC conferences started in 1987. Initially, EE task was limited to several domains and had small-sized corpora at its disposal. Nowadays, with the rapid evolution of socially-oriented Internet, it is becoming a crucial task for businesses to analyze millions of documents with multilingual content each day with minimal latency in order to get insight and provide critical decisions in time. We believe that modern and effective solutions to EE task would allow businesses to dramatically minimize the time required to start making use of new sources of information and reduce the operating costs of such systems.

One of the popular approaches to information extraction (IE), and EE as a sub-problem of IE in particular, is in the use of domain specific extraction patterns such as linear extraction patterns for annotation graphs and patterns for dependency-based parse tree (DPT) structures. In this paper we present our approach to semi-automatic generation of linear extraction patterns for annotation graphs while using DPT patterns in the process of constructing the first ones.

The paper is structured as follows. In the next section we briefly outline related work, in particular, we give an overview of the papers that propose algorithms minimizing human efforts required to build extraction patterns. In section 3 we describe in details four steps of the algorithm. Section 4 presents the results of experiments. We conclude in section 5 by summarizing the results and outlining current and future work.

2 Related Work

One of the most known algorithms that learn multi-slot extraction patterns for free texts is Whisk [10]. It requires an annotated corpus of documents and syntactic parse data provided with it.

The problem of learning conventional character-based regular expressions from annotated corpora is described in [8]. The authors demonstrate the applicability of their method in extracting such structured entities as software names, URLs, phone and course numbers. It is still an open question if their approach can be effectively applied for extracting higher level structures such as relations or events.

Snowball [1, 3, 9] and ExDisco [3, 11] are the examples of algorithms for learning information extraction patterns from unannotated corpora. Snowball is a logical successor of DIRPE [4, 1, 3, 9] which was intended for extracting binary relations from semi-structured web data. Snowball also extracts binary relations but it is intended for free texts. The main idea is that there is a small seed of known relations and these relations are to be found in the text. After that the algorithm generalizes sentence context (creates a pattern). The next step is to find new relations within this context.

ExDisco uses a different approach to deal with the problem of absence of annotated corpora. It divides a corpus into two parts (relevant and irrelevant documents) and uses the following assumption: relevant documents contain relevant events and relevant events are present in relevant documents. ExDisco uses a small seed of 2 or 3 known patterns to first divide documents, then it does the syntactic analysis of every sentence and uses its results for pattern generation.

Proceedings of the Ninth Spring Researcher's Colloquium on Database and Information Systems, Kazan, Russia, 2013

As opposed to conventional IE from relatively small annotated corpora, Open Information Extraction (OIE) deals with large Web-sized unannotated sets of documents. The two most known information extraction systems that do OIE are KnowItAll [6, 5, 3] and TextRunner [12, 5, 3]. They rely upon syntactic information rather than annotated or any lexical data.

3 Generating Linear Patterns

The motivation behind this research is to minimize human efforts in constructing event extraction patterns for new types of events. There are three assumptions underlying our algorithm. The first one is the way how we define the verity of events. We define an event to be a true (positive) event if its indicator and arguments linguistically represent a valid event mention no matter what polarity, modality etc. the event mention has. For instance, in the sentence "Reuters announces that HP will not acquire Palm" the pair {Reuters, announces} represents a valid event mention of the type *Company Announcement*, while {HP, announces} pair does not. On the other hand, the triple {HP, Palm, will not acquire} represents a valid event mention of the type *M&A*, while {Palm, Reuters, will not acquire} triple does not.

The algorithm requires the documents to be annotated with named entities that might be the arguments of events. They include not only such named entities as persons, companies, positions and event indicators, but also temporal and monetary expressions. A complete set of required named entities depends on the types of events to be extracted. We use available NERs (in particular, we use OpenNLP library) which we trust. It means that we consider texts annotated by appropriate NERs as the gold standard and we do not consider confidence of extracted entities (if NERs provide such information) during the process of evaluating the extraction patterns.

We also assume that if two events are extracted by the same DPT pattern, they both are either true or false. More generally, a group of *N* events all extracted by the same DPT pattern contains only either true or false events. A verity of the group is determined by the verity of a random sample drawn from it. This is the main assumption that allows us to minimize human efforts required to produce extraction rules by validating only such a small random sample but not the whole entire group that might contain tens of thousands of events.

As we have mentioned, the algorithm is composed of four steps. At the first step we extract possible events from an unannotated corpus of documents.

3.1 Extracting Possible Events

Any event type is described by an indicator (a word that most clearly signals about the event presence, usually, a verb) and arguments together with their semantic types. For instance, an event of the type *M&A* is described by an event indicator of the type *AcquisitionIndicator* and two arguments: *acquirer* of the type *Company* and *acquiree* of the type *Company* as well. Another example is an event of the type *PersonAnnouncement* that is described by an indicator of the type *AnnouncementIndicator* and one argument *announcer* of the type *Person*. Formally, an event is defined as the following triple:

$$E\left[T, I, \{A_i^N, A_i^T\}_{i=1}^m\right] \quad (1)$$

where *T* is the event type, *I* is the type of event indicator,the event has *m* arguments and the *i*-th argument has a name A_i^N and type A_i^T.

The core idea at the first step is to generate all possible events for every sentence in the corpus. To do it, we have built the processing pipeline presented at fig. 1. The primary task of this pipeline is to recognize instances of event indicators and named entities that might be arguments of events.

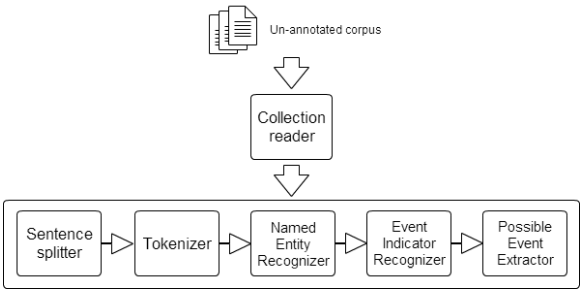

Figure 1: Pipeline that extracts possible events.

The pipeline has a typical architecture intended for extracting named entities. Initially, it splits text into sentences and tokens. After that, it uses Named Entity and Event Indicator Recognizers to extract named entities and event indicators. We use OpenNLP library to extract named entities and a dictionary-based recognizer to extract event indicators.

The last component of the pipeline (Possible Event Extractor) uses previously extracted information together with the description of events in order to extract possible events. First, this component determines if a sentence may contain at least one event of any type. To do it, it iterates over the description of event types. For each event type, it determines if a sentence contains (1) an indicator of the appropriate type, and if so (2) a minimal number of the appropriate named entities. For instance, for the event of the type *PersonAnnouncement* a sentence must contain at least one indicator of the type *AnnouncementIndicator* and at least one named entity of the type *Person*. For the event of the type *M&A*, a sentence must contain at least one indicator of the type *AcquisitionIndicator* and at least two named entities of the type *Company*. If a sentence cannot contain a mention of at least one event, it is not processed further.

If there can be at least one event mention in the sentence, we apply the Stanford parser to obtain DBT. For every type of the event *T*, for which there can be at least one event mention, we generate all possible events. At the first step, we construct a list of all appropriate event indicators $\{I\}_{i=1}^n$ found in the sentence. Then we construct a set of named entities that will be a part of possible events. We compute the shortest paths in DPT from indicators to every named entity in this set. We then generate all possible events around each event indicator. For every possible event, we construct its pattern id - a non unique string that is composed of the event type information and DPT paths. Two

events of the same type having the same paths from indicators to arguments will have the same pattern id. The format of the pattern id is the following: *Indicator* $Attribute_1Type\ (CoveredText_1):Path_1$, $Attribute_2Type\ (CoveredText_2):Path_2$, ... For instance, the event mentioned in the sentence "Google acquires Neotonic Software" has the pattern id *AcquisitionIndicator Company(Google):nsubj, Company(Neotonic Software):dobj*.

It is possible to count the number *N* of the events that are generated around each indicator. Let us denote the number of distinct types of arguments as $m/$, and for each *j*-th distinct type let S_j be the number of arguments in the event definition of this type, and let N_j be the number of named entities of this type in a sentence. Then there can be $P_j(N_j, S_j) = N_j!/(N_j - S_j)!$ variants to fill S_j arguments with N_j named entities. To compute the number of possible events, we need to multiply all these values $N = \prod_{j=1}^{m/} P_j(N_j, S_j)$.

Due to limitations of the Stanford parser, only those sentences that have less than 80 tokens are processed.

3.2 Grouping Possible Events into Clusters

Once the input corpus has been processed and all possible events have been identified, at the second step we group the events according to their pattern id. All events inside every group have the same pattern id. In other words, they are all extracted by the same DPT pattern. We sort the groups by cardinality and generate a random subset for each group.

3.3 Assessor Validation

At the third step an assessor validates a small random subset of each cluster using UIMA Cas editor[1] inside Eclipse IDE. This editor (fig. 2) highlights text fragments that mention possible events. For every possible event, this tool provides detailed information such as its indicator and arguments that are also highlighted in the text. The assessor should iterate through every possible event, decide if a particular event is true or false and in case it is true, set its verity flag to true.

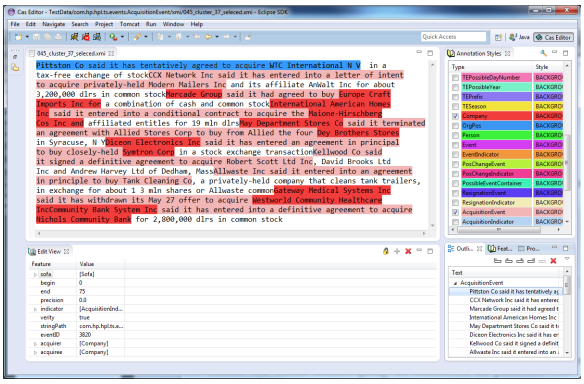

Figure 2: UIMA CAS Editor perspective (Eclipse IDE).

[1] http://uima.apache.org/

3.4 Generating Linear Patterns

Finally, the system annotates the groups of events as positive or negative based on the subsets validated by the assessor at the previous step. This is done by counting the number of true and the number of false events in the subset. If there are more positive events in the subset, the whole group is annotated as positive, and vice versa. Positive groups are used further to generate and generalize event extraction patterns.

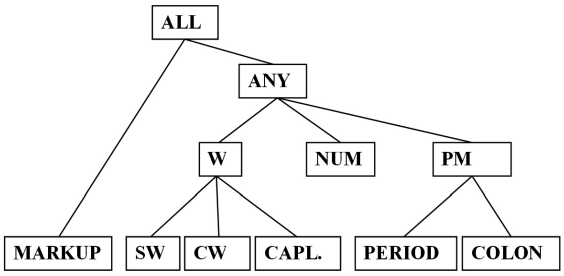

Figure 3: Typesystem of TextMARKER rule engine.

The input data for the pattern generation algorithm is the sentence annotation graph. The graph contains such annotations as events, their indicators and arguments (named entities) and sentence context which is presented as types of tokens in the notation of the TextMARKER[7] type system (fig. 3). We use the bottom-up generalization strategy and start with the most specific version of a pattern. We generalize patterns until the F1-measure of the new ones increases.

```
1:  procedure GeneralizeRule(rule)
2:    cur_F1 ← 0
3:    new_F1 ← TestRule(rule)
4:    while (new_F1 − cur_F1) > 0 do
5:      cur_F1 ← new_F1
6:      rules ← empty_list
7:      rules.add(BestRule(BottomUpGen(rule)))
8:      rules.add(BestRule(QuantModif(rule)))
9:      if NewRulesUnknown(rules) then
10:       newRule ← BestRule(rules)
11:       new_F1 ← TestRule(newRule)
12:       rule ← newRule
13:     end if
14:   end while
15:   return (rule)
16: end procedure
```

Figure 4: Pattern generalization algorithm.

There are two operations that we use during pattern generalizing process (fig. 4):

1. *BottomUpGen*: bottom-up generalization (see fig. 3). After the type of token is changed, its neighbors are visited. If there are two tokens with the same type close to each other, they are generalized to one token of the same type with an expanded quantifier: "company" "based" "in" "Tel" "Aviv" → SW SW SW CW CW → SW[3] CW[2] → W[5]

   (a) generalizing a word to its type: "recently" → SW; "May" → CW; "INC" → CAP

(b) generalizing the type of a word to its immediate parent type:
SW, CW, CAP → W

(c) generalizing the types of punctuation marks to their parent type:
COLON, COMMA, SEMICOLON → PM

(d) generalizing the types to their parent types:
W, PM → ANY

2. *QuantModif*: quantifier modification (expansion/restriction of a quantifier)

(a) Expansion: SW[3] → SW[2,4]

(b) Restriction: CW[2,7] → CW[2,6]

Using these two operations, we iteratively generalize the patterns. At every step we select the best pattern and we stop the generalizing process when the F1-measure stops improving (new_{F1}). We keep a list of previously generalized patterns. If the current pattern has already been processed, it is not generalized further. This corresponds to line 9 of the algorithm (fig. 4).

4 Experiments

Sentences annotated by the assessor with true events are considered as the gold standard and are used to measure the performance of event extraction patterns. Once the generated patterns had been applied over the gold standard, we compared the gold event annotations with the event annotations created by the patterns under evaluation. Two event annotations are considered to be the same if they cover the same event indicators and arguments (named entities).

We ran preliminary experiments on the English part of Reuters (RCV2)[2] dataset [2]. Table 1 presents the details of the entire dataset that we used (after processing all English articles on the pipeline that detects possible events). We ran experiments on extracting three types of events -

Table 1: Dataset description

type of event	M&A	MPC	Res
number of sentences	83	45	80
number of true events	12	16	25

Mergers & Acquisitions (M&A), *Management Position Change* (MPC) and *Resignation* (Res). We compare the performance of automatically constructed patterns with the patterns that we built manually based on RSS articles obtained from such sources as Yahoo and Google news feeds. In total, we manually created 10 patterns for *M&A*, 3 patterns for *MPC* and 5 patterns for *Res* events.

In the first experiment, we created 2 train/test splits. In each split, we constructed rules on the train set and validated them on the test one. Table 2 provides split details as well as the number of patterns that have been constructed automatically in each case. Tables 3-5 give an overview of the results of experiments. For *M&A* event, automatically constructed patterns outperformed manually created ones in terms of precision, recall and F1-measure on both the train and test sets in all splits (table 3), thought the number of automatically constructed patterns (8) is smaller than the number of manually created ones (10). Note that in the first split on the test data and in the second split on the train data manually constructed patterns did not extract any events, that is why we do not provide the performance for these cases.

[2]Initially, we wanted to use RCV1 that is much bigger. However, due to performance characteristics of the first implementations of the algorithm in terms of execution time, we decided to user smaller corpus and experiment on RCV1 after we optimize the algorithm.

Table 2: Splits of the entire dataset (train/test (patterns#))

	1st split	2nd split
M&A	61/22 (8)	55/28 (8)
MPC	38/7 (9)	30/15 (8)
Res	54/26 (10)	56/24 (10)

Table 3: *M&A* patterns performance (train / test)

		Manually	Automatically
1st split	Precision	0.50 / —	1.00 / **1.00**
	Recall	0.11 / 0.00	1.00 / **0.33**
	F1-measure	0.18 / —	1.00 / **0.50**
2nd split	Precision	— / 0.50	1.00 / **1.00**
	Recall	0.00 / **0.50**	1.00 / **0.50**
	F1-measure	— / 0.50	1.00 / **0.66**

In case of *MPC* events, in the first split we got higher results using automatically constructed patterns (see table 4). In the second split, manually and automatically constructed patterns demonstrated the same results, however, our algorithm constructed 8 patterns as opposed to 3 patterns constructed by a human.

Table 4: *MPC* patterns performance (train / test)

		Manually	Automatically
1st split	Precision	0.78 / 0.50	0.92 / **1.00**
	Recall	0.58 / 0.33	1.00 / **0.33**
	F1-measure	0.67 / 0.40	0.96 / **0.50**
2nd split	Precision	0.67 / **1.00**	1.00 / **1.00**
	Recall	0.60 / **0.40**	1.00 / **0.40**
	F1-measure	0.63 / **0.57**	1.00 / **0.57**

Automatically constructed patterns for *Res* events also outperformed those manually created in terms of F1-measure in the first and second splits. However, the precision of manually constructed patterns was higher in both splits.

In the second experiment, we tried to get more realistic comparison results of manually and automatically constructed patterns. What we did was validation of manually constructed patterns on the entire dataset (table 7) and ran the algorithm through a 5-fold cross validation 5 times (due to the fact that we used a relatively small dataset). The results are presented in table 6.

Afterwards, we tested manually constructed patterns on the entire dataset. The results are given in table 7. As it can be seen, we got quite varying results for different partitions (F1-measure: 0.4523-0.8333 for *M&A*, 0.5428-0.7023 for *MPC* and 0.5000- 0.6764 for *Res* events). A conclusion that can be made from these results is that we need a larger and more representative dataset to construct and evaluate comprehensive event extraction patterns.

Table 5: *Res* patterns performance (train / test)

		Manually	Automatically
1st split	Precision	1.00 / **1.00**	0.83 / 0.50
	Recall	0.18 / 0.33	0.86 / **0.67**
	F1-measure	0.30 / 0.50	0.84 / **0.57**
2nd split	Precision	1.00 / **1.00**	0.90 / 0.33
	Recall	0.19 / 0.25	0.90 / **0.75**
	F1-measure	0.32 / 0.40	0.90 / **0.46**

Table 6: F1-measure of automatically constructed rules (5-fold cross validation 5 times)

M&A	MPC	Res
0.4523	0.5428	0.5000
0.4666	0.5714	0.5988
0.4888	0.6750	0.6321
0.5555	0.6904	0.6426
0.8333	0.7023	0.6764

5 Conclusions and Future Work

In this paper we gave an overview of the current progress in developing the algorithm for semi-automatic generation of linear event extraction patterns for free texts and presented our preliminary experimental results. Our next steps are to enhance the algorithm with additional capabilities, optimize it in terms of execution performance and validate it on a larger dataset.

Currently, we extract only mandatory events' arguments. We will add capability to generalize patterns that will include optional arguments as well (such as temporal and monetary expressions). We plan to explore the possibility to improve and optimize the algorithm. We will work on enhancing the generalization algorithm by providing additional operations as well as an intelligent selection of operations to apply at each iteration. We will also explore the possibility to employ negative examples. We have a dataset of approximately 110 000 news articles that we will use for validating the algorithm. We will also use Reuters RCV1 dataset for this purpose. We will study in much more details theoretical properties of the proposed algorithm.

Table 7: Performance of manually constructed rules on the entire dataset

	M&A	MPC	Res
Precision	0.5	0.73	1.0
Recall	0.08	0.53	0.2
F1-measure	0.14	0.62	0.33

References

[1] Eugene Agichtein and Luis Gravano. Snowball: extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*, DL '00, pages 85–94, New York, NY, USA, 2000. ACM.

[2] Massih-Reza Amini, Nicolas Usunier, and Cyril Goutte. Learning from multiple partially observed views - an application to multilingual text categorization. In *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, pages 28–36, 2009.

[3] Nguyen Bach and Sameer Badaskar. A Review of Relation Extraction. 2007.

[4] Sergey Brin. Extracting patterns and relations from the world wide web. In *Selected papers from the International Workshop on The World Wide Web and Databases*, WebDB '98, pages 172–183, London, UK, UK, 1999. Springer-Verlag.

[5] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld. Open information extraction from the web. *Commun. ACM*, 51(12):68–74, December 2008.

[6] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 100–110, New York, NY, USA, 2004. ACM.

[7] Peter Kluegl, Martin Atzmueller, and Frank Puppe. Integrating the rule-based ie component textmarker into uima. In Joachim Baumeister and Martin Atzmueller, editors, *LWA-2008 (Special Track on Information Retrieval)*, pages 73–77, 2008.

[8] Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and H. V. Jagadish. Regular expression learning for information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 21–30, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[9] Ryan McDonald. Extracting relations from unstructured text. Technical report, 2005.

[10] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, 34(1-3):233–272, February 1999.

[11] Roman Yangarber, Ralph Grishman, and Pasi Tapanainen. Automatic acquisition of domain knowledge for information extraction. In *In Proceedings of the 18th International Conference on Computational Linguistics*, pages 940–946, 2000.

[12] Alexander Yates, Michele Banko, Matthew Broadhead, Michael J. Cafarella, Oren Etzioni, and Stephen Soderland. Textrunner: Open information extraction on the web. In *HLT-NAACL (Demonstrations)'07*, pages 25–26, 2007.