Rapid Customization for Event Extraction

Yee Seng Chan, Joshua Fasching, Haoling Qiu, and **Bonan Min**
Raytheon BBN Technologies, Cambridge, Massachusetts
{yeeseng.chan,joshua.fasching}@raytheon.com
{haoling.qiu,bonan.min}@raytheon.com

Abstract

Extracting events in the form of *who* is involved in *what* at *when* and *where* from text, is one of the core information extraction tasks that has many applications such as web search and question answering. We present a system for rapidly customizing event extraction capability to find new event types (*what* happened) and their arguments (*who, when, and where*). To enable extracting events of new types, we develop a novel approach to allow a user to find, expand and filter event triggers by exploring an unannotated development corpus. The system will then generate mention-level event annotation automatically and train a neural network model for finding the corresponding events. To enable extracting arguments for new event types, the system makes novel use of the ACE annotation dataset to train a generic argument attachment model for extracting *Actor*, *Place*, and *Time*. We demonstrate that with less than 10 minutes of human effort per event type, the system achieves good performance for 67 novel event types. Experiments also show that the generic argument attachment model performs well on the novel event types. Our system (code, UI, documentation, demonstration video) is released as open source.[1]

1 Introduction

Event extraction is the task of identifying events of interest with associated participating arguments in text. For instance, given the following sentence:

S1: 21 people were wounded in Tuesday's southern Philippines airport blast.

Event extraction aims to recognize the two events (Injury and Attack), *triggered* by the words "wounded" and "blast" respectively. We also recognize that "21 people" and "airport" take on the event argument roles *Actor(s)* involved and *Place* respectively.

[1] github.com/BBN-E/Rapid-customization-events-acl19

For event trigger and argument extraction, state-of-the-art approaches employ supervised machine learning methods. These methods assume a predefined event ontology and learn from a corpus of manually labeled examples that are specific to that ontology. For instance, the popular Automatic Content Extraction (ACE) (Doddington et al., 2004) corpus contains 599 documents manually annotated with examples for 33 event types, such as *Attack* and *Justice* events.

However, producing such type-specific examples is labor intensive. To extract triggers and arguments of a new event type, one needs to annotate a large amount of training examples specific to that new event type. For instance, ACE provides event-type specific argument annotations, such as *Attacker* for *Attack* events. This prevents existing event argument examples from being useful towards extracting participants of new event types, as initially defined.

In this paper, we present a system that facilitates rapid extension of extraction capabilities to a large number of novel event types. We summarize the contributions of this paper as follows:

- We present an approach to rapidly gather event trigger examples for new event types, with minimal human effort.
- We develop a User Interface (UI) to further expedite and improve the time efficiency of our approach.
- For event arguments, we show how to leverage annotations of $existing$ event types and argument roles, to train a classifier that extracts event arguments such as *Actor* (*who* is involved), *Place* (*where* it happened) and *Time* (*when* it happened) for the new event types.
- We demonstrate the practical utility of our approach by applying it on a set of 67 novel event types.

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pages 31–36
Florence, Italy, July 28 - August 2, 2019. ©2019 Association for Computational Linguistics

Figure 1: A user interface that allows a user to provide, expand, and filter event triggers for new types. A demonstration video is available on github.com/BBN-E/Rapid-customization-events-acl19.

We first describe the task of event extraction in the next section. In Section 3, we describe our extraction model, how we leverage our UI to rapidly gather event trigger examples for new event types, and how to extract event arguments for new event types. We present experiment results in Section 4. We discuss related work in Section 5 before concluding in Section 6.

2 Problem Definition

We focus on the problem of *rapid customization of event extractors* for new event types where we don't have a large amount of hand-labeled data available. Given an English sentence, we perform event extraction using a two-stage process:

- Trigger classification: Labeling words in the sentence with their predicted event type (if any). For instance, in sentence S1, the extraction system should label "wounded" as a trigger of an Injury event, and label "blast" as a trigger of an Attack event.
- Argument classification: If a sentence contains predicted event triggers $\{t_i\}$, we pair each t_i with each entity and time mention $\{m_j\}$ in the sentence to generate candidate event arguments. Given a candidate event argument (t_i, m_j), the system predicts its associated event role (if any). For instance, given ("wounded", "airport"), the system should predict the event role $Place$.

3 Approach

3.1 A Convolutional Neural Network Model for Event Extraction

We developed a convolution neural network (CNN) model to perform event trigger classification, and another CNN model for event argument classification used with our novel trigger and argument example collection approaches. Both CNN models are very similar, with the argument model incorporating more features. Hence, we will describe the argument model in detail, then provide a summary of the trigger model.

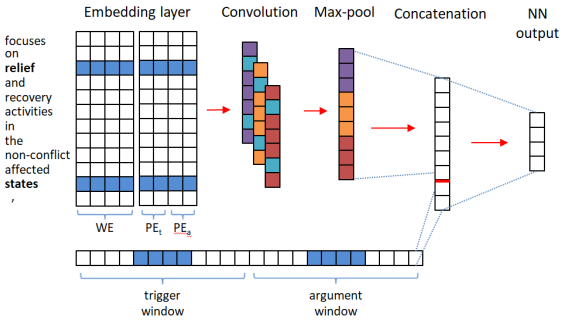

Figure 2: A CNN based model for event argument classification. WE is word embeddings. PE_t and PE_a are position embeddings, capturing a token's distance to the candidate trigger and argument respectively. These position embeddings are randomly initialized and learnt during training.

As shown in Figure 2, the argument model consists of (1) an embedding layer to encode words and word positions in the sentence, (2) a convolution and max pooling layer to generate high-level features from the embedding representation of the sentence, (3) a layer which concatenates the max pool layer and local context window around the candidate trigger and argument, (4) followed by the softmax function for classifying the example into one of the target classes.

For argument classification, the input is a sentence in which a trigger word and a candidate event argument is identified, e.g. ("relief", "states") in Figure 2.

Embedding Layer encodes each word with:

- Word embeddings (WE): Given an input sentence $\mathbf{x}$ of length t, we first transform each word into a real-valued vector of dimension d_1 by looking up a word embedding matrix $W^1 \in R^{d_1 \times |V|}$, where V is the vocabulary. We use word embeddings trained by Baroni

et al. (2014), which achieved state-of-the-art results in a variety of NLP tasks.

- Position embeddings (PE): PE_t encodes the relative distance of each word to the trigger word as a real-valued vector of dimension d_2 by a embedding matrix $W^2 \in R^{d_2 \times |D|}$, where D is the set of relative distances in a dataset. W^2 is randomly initialized and learnt during training. We similarly use PE_a to encode relative distances to the candidate argument, by defining $W^3 \in R^{d_3 \times |D|}$.

The final embedding dimension for each token is $n_1 = (d_1 + d_2 + d_3)$. This layer produces an embedding representation $\mathbf{x}^{(1)} \in R^{n_1 \times t}$ when fed with an input sentence $\mathbf{x}^{(0)} = \mathbf{x}$.

Convolution and Max Pooling Layer: We use a set of filters with different window sizes to capture important n-gram features from an input sentence. Due to space constraints, we omit the definitions of the convolution and max pool layers. We denote the max pool layer using a fixed-sized feature vector $\mathbf{x}^{(2)} \in R^{n_2}$, where n_2 is the total number of filters.

Concatenate Layer: We select the word embeddings of the trigger, the candidate argument, and their local windows. We define the window surrounding a word, as the *k=3* tokens to the left and right of the word. We concatenate these embeddings to the max pool layer, to obtain a concatenated vector $\mathbf{x}^{(3)}$.

Event Argument Classification: We have $\mathbf{o} = W^{(3)}\mathbf{x}^{(3)} + \mathbf{b}^{(3)}$, where $W^{(3)}$ and $\mathbf{b}^{(3)}$ are parameters learnt in this layer. Here, $\mathbf{o} \in R^{n_3}$, where n_3 is equal to the number of event argument roles including the "NONE" label for candidate arguments which are not actual event arguments to the trigger. Given an input example $\mathbf{x}$, our network with parameters θ outputs the vector $\mathbf{o}$, where the i-th component contains the score for event role i. To obtain the conditional probability $p(i|\mathbf{x}, \theta)$, we apply softmax:

$$p(i|\mathbf{x}, \theta) = \frac{e^{o_i}}{\sum_j e^{o_j}} \quad (1)$$

The CNN for trigger classification is largely the same as the above CNN for argument classification, omitting just the argument associated features, i.e. PE_a and the argument window shown at the bottom of Figure 2. The input is a sentence in which a word is the candidate trigger word, e.g. "relief" in Figure 2. The output is a softmax function predicting one of the event type or *NONE*, indicating the candidate word is not a valid trigger for any of the event types.

3.2 Rapid Customization for Event Trigger Extraction

Our system enables rapidly gathering of event trigger examples for new event types with minimal human effort, aided by the UI shown in Figure 1, using this work flow:

- Given a new target event type, the user first provides some initial keywords. The UI (backed by an unannotated text corpus) presents up to 3 text snippets (sentences) mentioning each trigger.
- The user can then easily gather additional discriminative keywords using the UI via interactive search. By clicking on the "Find similar" button in each pane, the system will suggest new event keywords that are similar to the current set of keywords, displaying these suggested keywords in the working pane on the left of the UI. Our system suggests new keywords using WordNet hyponyms and cosine similarity in a word embedding space.
- The user can then repeat this process for additional event types. This can be seen in Figure 1, where each pane (column) shows an event type name at the top, followed by event triggers (in red) and text snippets (clickable to expand to full sentence) mentioning these triggers.
- The user can edit between event types by drag and drop, moving a trigger or snippet from one event to another. The user can also click on "−" to remove an event, a trigger with its snippets, or just a snippet. The user can also click on the "More" button to the right of each trigger, to display additional text snippets containing the trigger.
- When the user is satisfied with the current set of keywords and associated text snippets, our system then performs distant supervision (Mintz et al., 2009) by using the occurrences of these keywords (their associated text snippets) as event trigger examples for the new event type.

In practice, over a set of 67 new event types described in Section 4.2, the user spent an average of 4.5 minutes to provide 8.6 initial triggers and associated text snippets. Then another 5 minutes inter-

acting with the UI to expand and filter the triggers, for a total of less than 10 minutes per event type.

3.3 Argument Extraction for New Events

Current argument examples, such as those defined in ACE, are event type specific. For instance, the ACE corpus annotates Agent and Victim arguments for Injure events, Attacker and Target arguments for Attack events, etc. To decode event arguments for new event types, one need to annotate new event type specific argument examples as training data.

In this paper, we propose a simple approach to learn a *generic* event argument model to extract Actor, Place, and Time arguments for *any new* event types, without annotating new examples. We define Actor as a coarse-grained event argument role, encompassing Agent-like and Patient-like event roles. We map Actor-*like* argument roles in ACE to a common Actor role label, and use the Place and Time arguments in as they appear in ACE. The complete list of ACE event argument roles that we mapped to Actor are:

- Agent, Artifact, Adjudicator, Victim, Buyer, Seller, Giver, Recipient, Org, Attacker, Target, Entity, Defendant, Person, Plaintiff, Prosecutor

Using the above mapping approach, we train a *generic* event argument classifier that can extract Actor, Place, and Time arguments for any event type.

4 Experiments

4.1 Verifying Event Extraction Model

We first conduct experiments to verify that our CNN model implementation achieves comparable performance to state-of-the-art CNN-based event extraction systems (Chen et al., 2015; Boros, 2018) to ensure that it is suitable for use in our rapid event customization approach. Following these prior work, we use the ACE-2005 corpus, with the same sets of 529 training documents, 30 development documents, and 40 test documents. We use the same following criteria to judge the correctness of our event extractions: A trigger is correctly classified if its event subtype and offsets match those of a reference trigger; an argument is correct classified if its event subtype, event argument role, and offsets match any of the reference event arguments.

	$\mathcal{C}_{ds}$	$\mathcal{C}_{adj}$	$\mathcal{C}_{ds'}$	Dev	Test
#articles	818	618	618	274	273
#triggers	1674	1171	1258	643	752

Table 1: Counts of articles and trigger examples, in training corpora for distant supervision (C_{ds}), distant supervision followed by human adjudication (C_{adj}), and sampled distant supervision ($C_{ds'}$), as well as corpora for development (Dev) and test (Test).

Since the ratio of positive (valid) vs negative examples is relatively skewed (for instance, most words in a sentence are not triggers), we tried different weights for the positive examples: 1, 3, 5, or 10. We tune this and other hyper-parameters (batch size, number of CNN filters, number of epoches) on the development documents. We also follow (Chen et al., 2015) by using the Adadelta update rule with parameters $\rho = 0.95$ and $\epsilon = 1e^{-6}$, and a dropout rate of 0.5. On the ACE test data, our trigger model achieves an F1 score of 0.65, close to the scores of 0.66 and 0.68 reported in (Chen et al., 2015) and (Boros, 2018) respectively. Our argument model using gold triggers[2] achieves an F1 score of 0.53, close to the score of 0.55 reported in (Boros, 2018).

4.2 Event Customization Evaluation

To evaluate the effectiveness of our event extraction system in customizing extractors for new event types, we present experiment results based on the Common Core Ontologies[3] (CCO). CCO comprises 11 ontologies and is aimed at representing semantics for many domains of interests. We sampled 67 event types that are not in existing event schemas (such as ACE and TAC-KBP[4]), to evaluate how well our system does on novel event types. As our experiment corpus $\mathcal{C}$, we use 6,000 allafrica.com news articles, published between 2016-2017.

4.2.1 Trigger Classification

Given the set of 67 new event types, we leverage our UI to obtain a set of keywords that are associated with about 3,000 trigger examples spanning 1,365 articles. We split these examples at the article level via a 60/20/20 train/development/test split. We show the statistics of our data in Table 1. We then trained the following models:

- We trained a trigger model T_{ds} using the 1,674 training examples $\mathcal{C}_{ds}$. Note that $\mathcal{C}_{ds}$

[2]Since comparisons using predicted triggers obfuscate event argument performance.

[3]https://github.com/CommonCoreOntology

[4]https://tac.nist.gov/2017/KBP/Event/index.html

Type	Triggers
Ceremony	celebration, ceremony, parade, commemoration, feast, ...
Criminal Act	abduction, assassin, assault, bandit, blackmail, bribery, ...
Cyber Attack	botnet, cyber attack, cyber war, cyber warfare, cybercrime, ...
Espionage	espionage, infiltrate, infiltrator, mole, saboteur, spy

Table 2: Sample triggers for some CCO event types.

	Precision	Recall	F1
T_{ds}	0.69	0.50	0.58
T_{adj}	0.69	0.46	0.55
$T_{ds'}$	0.62	0.40	0.48

Table 3: Event trigger results on new CCO event types.

consists of distant supervised (DS) trigger examples which are potentially noisy.

- We adjudicated $\mathcal{C}_{ds}$, obtaining a smaller set of 1,171 trigger examples $\mathcal{C}_{adj}$, which we used to train a trigger model T_{adj}.

Table 2 shows examples of triggers identified by our in-house developer for sampled CCO events. When evaluated on the test examples, T_{ds} and T_{adj} achieve F1 scores of 0.58 and 0.55 respectively (shown in Table 3).

The impact of corpora size Surprisingly, the DS model T_{ds} (trained on the noisy distance supervised $\mathcal{C}_{ds}$) performs better than the model T_{adj} (trained on the manually adjudicated $\mathcal{C}_{adj}$). One possible explanation is because $\mathcal{C}_{adj}$ is a subset of $\mathcal{C}_{ds}$ and contains significantly fewer examples, since only trigger examples that are judged to be correct for the event types are kept.

Table 1 shows that $\mathcal{C}_{adj}$ contains substantially fewer examples than $\mathcal{C}_{ds}$ (1,171 vs 1,674). To verify that the larger number of training examples is a reason for T_{ds}'s higher performance, we randomly down-sampled $\mathcal{C}_{ds}$ to have the same number of documents as $\mathcal{C}_{adj}$. Using the resulting $\mathcal{C}_{ds'}$, we trained the trigger model $T_{ds'}$. When evaluated on the test data, this obtains a F1 score of 0.48, which is indeed worse than T_{adj} as expected, thus confirming our hypothesized explanation. We show these results in Table 3.

4.2.2 Argument Extraction

We apply the mapping approach described in Secton 3.3 on the ACE data. We learn a *generic* argument model A_{gen} on the mapped training data, obtaining a F1 score of 0.50 when evaluated on the mapped ACE test data (Table 4). For comparison,

Model	Overall			F1		
	P	R	F1	Actor	Place	Time
A_{gen}	0.65	0.41	0.50	0.49	0.37	0.61
A_{out}	0.41	0.62	0.49	0.49	0.45	0.62

Table 4: Event argument results using gold triggers.

we also trained a model using the original ACE event roles in the standard way, but report results after mapping predicted and reference roles to a common Actor role. We obtained a similar test F1 score of 0.50.

We note that A_{gen} trains on the entire ACE training data. However, the motivation for the mapping is to learn an argument model for decoding on new event types not previously seen in its training data. Hence, we conduct an additional set of leave-1-out experiments A_{out}. ACE defines event types at a coarse-grained (8 types) and a fine-grained (33 types) level. Hence in each fold i, we omit argument examples associated with a coarse-grained ACE event type i from training, then proceed to calculate performance on just argument test examples associated with event type i. We aggregate the test results over all folds in row A_{out} of Table 4. We note that A_{out} achieves reasonable performance when compared against A_{gen}, demonstrating the viability of our approach towards extracting event arguments for previously unseen new event types.

Using T_{ds} and A_{gen} as the trigger and argument models, we decoded on our CCO test data. Of a set of randomly selected 78 Actor, 8 Place, and 14 Time arguments predicted by A_{gen}, we determine that 62 Actor, 7 Place, and 10 Time arguments are correctly predicted, for an overall precision of 0.79.

5 Related Work

Recent event extraction work usually employ neural network (NN) models, such as CNN-based models (Chen et al., 2015; Boros, 2018) and joint event extraction using recurrent neural networks (Nguyen et al., 2016a).

In event extraction using limited training data, Nguyen et al. (2016b) proposed a two-stage NN model for event type extension. Given a new event type with a small set of seed examples, they leverage examples from other event types. In another work, Peng et al. (2016) developed a minimally supervised approach to event trigger extraction by leveraging trigger examples gathered from the ACE annotation guidelines. Ferrero et al. (2017) presented InToEventS, an interactive tool for building event schemas. Their work dif-

fers from ours in several important aspects. Their tool produces schemas (triggers and role patterns) of events based on clusters, whereas our tool allows users to rapidly produce event trigger examples. Our tool also allows these examples to be adjudicated, allows multiple event types to be examined in parallel in the same UI, and triggers (or snippets) to be shifted across different event types. Finally, we demonstrate a viable approach for extracting Actor, Place, and Time arguments of new event types without any additional annotation effort.

A closely related direction is rapid customization of systems for other information extraction (IE) tasks. The ICE system (He and Grishman, 2015) allows a user to interactively create new classes of entities and relations. The main ideas are user-in-the-loop entity set expansion and boostrap learning for relation extraction. The WIZIE (Li et al., 2012) system guides users to write rules for IE. Finally, Michael and Akbik (2015) and Freedman et al. (2011) presented systems for interactively building relation extractors.

6 Conclusion and Future Work

We presented a system which allows a user to rapidly build event extractors to find new types of events and their arguments. We plan to use clustering techniques to automatically discover salient event trigger words in a new corpus, to further reduce human customization effort.

7 Acknowledgements

This work was supported by DARPA/I2O and U.S. Army Research Office Contract No. W911NF-18-C-0003 under the World Modelers program. The views, opinions, and/or findings contained in this article are those of the author and should not be interpreted as representing the official views or policies, either expressed or implied, of the Department of Defense or the U.S. Government. This document does not contain technology or technical data controlled under either the U.S. International Traffic in Arms Regulations or the U.S. Export Administration Regulations.

References

Marco Baroni, Georgiana Dinu, and German Kruszewski. 2014. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL-2014*, pages 238–247.

Emanuela Boros. 2018. *Neural Methods for Event Extraction*. Ph.D. thesis, Universite Paris-Saclay.

Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. 2015. Event extraction via dynamic multi-pooling convolutional neural networks. In *ACL-IJCNLP2-2015*, pages 167–176.

George R. Doddington, Alexis Mitchell, Mark A. Przybocki, Lance A. Ramshaw, Stephanie M. Strassel, and Ralph M. Weischedel. 2004. The automatic content extraction (ace) program - tasks, data, and evaluation. In *LREC*.

German Ferrero, Audi Primadhanty, and Ariadna Quattoni. 2017. InToEventS: an interactive toolkit for discovering and building event schemas. In *EACL Software Demonstrations*.

Marjorie Freedman, Lance Ramshaw, Elizabeth Boschee, Ryan Gabbard, Gary Kratkiewicz, Nicolas Ward, and Ralph Weishedel. 2011. Extreme extraction: machine reading in a week. In *EMNLP*.

Yifan He and Ralph Grishman. 2015. ICE: Rapid Information Extraction Customization for NLP Novices. In *NAACL-D-2015*, pages 31–35.

Yunyao Li, Laura Chiticariu, Huahai Yang, Frederick Reiss, and Arnaldo Carreno-fuentes. 2012. Wizie: A best practices guided development environment for information extraction. In *ACL-SD-2012*, pages 109–114.

Thilo Michael and Alan Akbik. 2015. A web toolkit for exploratory relation extraction. In *ACL-IJCNLP System Demonstration*.

Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *ACL-IJCNLP*, pages 1003–1011.

Thien Huu Nguyen, Kyunghyun Cho, and Ralph Grishman. 2016a. Joint event extraction via recurrent neural networks. In *NAACL-HLT-2016*, pages 300–309.

Thien Huu Nguyen, Lisheng Fu, Kyunghyun Cho, and Ralph Grishman. 2016b. A two-stage approach for extending event detection to new types via neural networks. In *WRepL4NLP*, pages 158–165.

Haoruo Peng, Yangiu Song, and Dan Roth. 2016. Event detection and co-reference with minimal supervision. In *EMNLP-2016*.