Event Nugget Detection and Coreference Scoring

Aug 12, 2015
Language Technologies Institute
Carnegie Mellon University

1 Overall workflow

This document describes the event nugget and coreference evaluation. We show an overall workflow of evaluation in Figure 1.

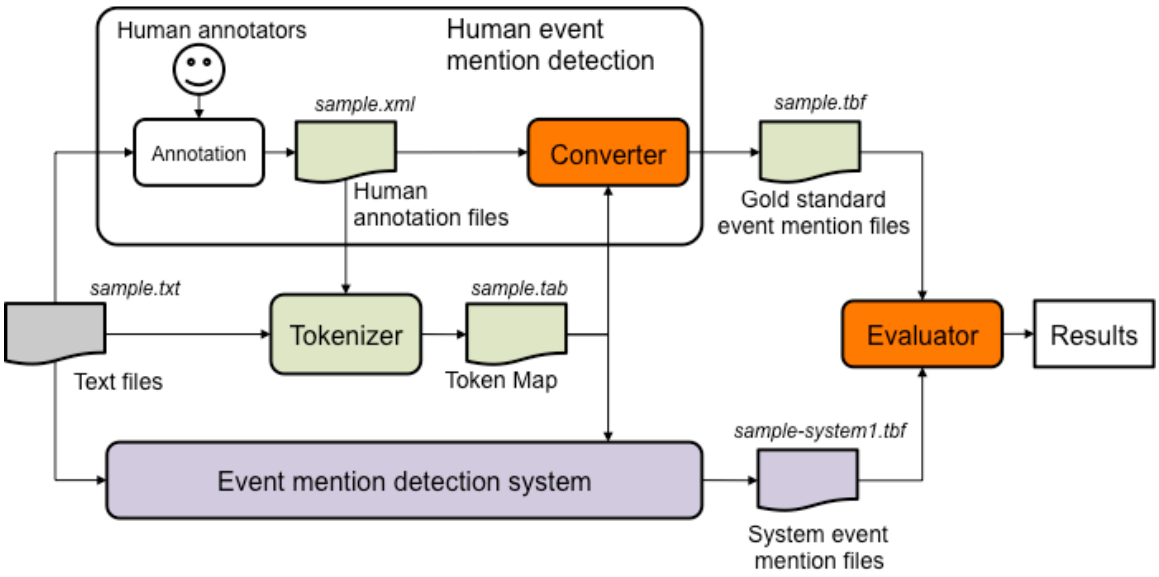

Figure 1: An overall workflow of evaluation for event nugget detection.

We first give an overview of the whole scoring process. For detailed format description please refer to the corresponding sections.

For each text file, LDC annotators provide human annotations as gold standard files. System submissions will be evaluated against certain gold standard based on the task it participates. Because existing evaluation for event nugget detection evaluation and event coreference evaluation all score systems on token level, we will conduct a post-tokenization step after the annotations. The post-tokenization step will create a token mapping file, which provide token offset and token id information.

Each participant event nugget detection system will be given two files as input: (1) the source text and (2) a token mapping table. The latter specifies token ID for every token. The ID information is used in the output of the system. Let us refer to the output of the system as a system event nugget file. We require a system event

nugget file to be given in the same file format as the gold standard file. The evaluator (scorer) takes the gold standard file, the system event nugget file, and the token mapping file as input, and compares them to give a score for the system.

2 Evaluation Process and Formats

The submission format described below will be used for both Event nugget Detection task and Event Coreference task.

Input of Scorer:

1. Gold standard annotation for documents, in Token Based Format (tbf).
2. System output annotation for documents submitted by participants, in Token Based Format (tbf)
3. Tokenization mapping files associated with each document, "tab" extension is appended to the file extension of its corresponding source filename.

Output of Scorer:

1. Overall performance report for system, as described in "Scoring" section.
2. Visualization in text form or html form if the corresponding arguments are specified. (This is not the core functionality for scoring, please refer to the README file of the scorer for details)

Evaluation Script Options:

Please follow the README and help information in the script distribution.

2.1 Formats

2.1.1 Tokenization file format

Standard token mapping files (.tab) will be provided to participants. These are tab-delimited mapping files for evaluation purpose. These files map the tokens to their offsets [1] in the source files. "tab" is appended to the file extension of its corresponding source filename. Participants should report their system output in terms of the token id provided in the token mapping files. These files will be provided, however, the generation process is described in the Appendix.

A mapping table contains 4 columns for each row, and the rows contain an ordered listing of the document's tokens. The columns are:

- token_id: A string of "t" followed by a token-number beginning at 0
- token_str: The literal string of a given-token
- tkn_begin: Index of the token's first character in the source file
- tkn_end: Index of the token's last character in the source file

2.1.2 The submission format

There are 3 possible tasks for evaluation this year, the submission format of these tasks are the same. The content of the submission may vary:

[1] Offsets are character offsets where the first character is 0

1. Event Nugget Evaluation: Only contain the event nugget annotations by the systems
2. Event Nugget Detection and Coreference: Contain both event nugget annotations and coreference by the systems
3. Event Nugget Coreference: Contain both event nugget annotations and coreference, **where event nugget annotations should always be (and must be) copied directly from provided event nugget files, coreference annotations are generated by the systems**[2]

The following sections describe the format in details.

2.1.3 System and gold standard annotation file format:

1. All event nugget annotations (and/or coreference annotation) for all documents in the corpus are written into one single file
2. A header will indicate the start of a new document (<s> is the space character)
    a. Header := #BeginOfDocument<s><doc ID>
3. A footer will indicate the end of a document
    a. Footer := #EndOfDocument

2.1.4 Definition of event nugget format:

Event nuggets are represented as tab-separated lines. To be specific, for each mention line, we have the following columns:

- <system ID> := the name of the system in order to distinguish participants
- <doc ID> := the ID of the input document
- <mention ID> := the ID of the mention, which should uniquely identify the mention within the current document
- <token ID list> := list of IDs for the token(s) of the current mention, in ascending order, separated by commas (,) . Each ID is a string of "t" followed by a token-number beginning at 0, the same as how they appear in the tokenization files
- <mention> := the actual character string of the mention
- <event-type> := the ACE hierarchy type
- <realis status> := the REALIS label

It is optional to append the following confidence columns (which can be helpful for follow up study and performance diagnostics):

- <span confidence> := a score (confidence, etc.) the system wants to assign for the mention span detection. This score will not affect the evaluation results
- <type confidence> := a score (confidence, etc.) the system wants to assign for the mention type detection. This score will not affect the evaluation results

[2] It is necessary to copy all event nugget annotations, even though they are provided because the scorer will simply read the response file. The scorer will complain if the an event mention in the chain is not contained in the list of nuggets provided.

- <realis confidence>:= a score (confidence, etc.) the system wants to assign for the mention realis detection. This score will not affect the evaluation results

2.1.5 Definition of event coreference format:

All coreference clusters annotations are **appended after all event nuggets lines of this file, before the #EndOfDocument footer**. System submissions should make sure transitive closure of coreference are resolved so that each coreference line indicate the whole cluster.

Each coreference cluster should also be represented as a tab-separated line, with the following columns:

- <relation name> := The relation name with a special indicator character (@) as prefix , **in this task, it is always "@Coreference"**
- <relation id> := A relation id. This is for bookkeeping purposes, which will not be read by the scorer. The relation id used in the gold standard files will be in form of "R<id>" (e.g. R3). However, system should **always** give a non-empty string without whitespace here. We recommend putting the cluster id as a placeholder.
- <event mentions> := A list of event mention ids in this coreference cluster, separated by comma (,). In terms of coreference, the ordering of event mentions does not matter. Each event mention id should have already presented in the list of nugget definition above.

Example:

```
#BeginOfDocument sample
system1    sample    E2    t1069  married    Life_Marry    Actual 1 1 1
system1    sample    E4    t1096  divorced   Life_Divorce  Actual 1 1 1
system1    sample    E5    t1109  married    Life_Marry    Actual 1 1 1
system1    sample    E6    t1157  married    Life_Marry    Actual 1 1 1
@Coreference    R1    E6,E2
#EndOfDocument
```

3 Event Nugget Detection Scoring

In this section we describe the algorithm for scoring event nugget detection.

3.1 Scoring for one document

We denote a gold standard mention with **G**, and a system mention with **S**. We use $\mathbf{T_S}$ to represent the tokens of the mention. $\mathbf{Dice(T_G,T_S)}$ is the token-based dice coefficient function that returns a score between 0 and 1 (All invisible words are already removed from $\mathbf{T_G}$ and $\mathbf{T_S}$ [3]).

[3] Invisible words are ignored in scoring. They include: determiners {the, a, an}, pronouns {I, you, he, she, we, my, your, her, our}, relative pronouns {who, what, where, when}.

3.1.1 Create mappings

To perform scoring for a document, system mentions are mapped to gold standard mentions based on the Overlap score (computed as token level Dice Coefficient). We use a greedy algorithm for comparing such the score:

Input: A list L of scores $Dice(T_G, T_S)$ for all pair of G, S in the document

1: $M \leftarrow \varnothing$; $U_s \leftarrow \varnothing$; $U_g \leftarrow \varnothing$;
2: **while** $L \neq \varnothing$ **do**
3: $\quad G_m, S_n \leftarrow \arg\max_{(G,S)\in L} Dice(T_G, T_S)$
4: $\quad L \leftarrow L - \{Dice(T_{G_m}, T_{S_n})\}$
5: $\quad$ **if** $S_n \notin U_s$ **and** $G_m \notin U_g$ **and** $Dice(T_{G_m}, T_{S_n}) > 0$ **then**
6: $\quad\quad M_{G_m} \leftarrow (S_n, Dice(T_{G_m}, T_{S_n}))$
7: $\quad\quad U_s \leftarrow U_s \cup \{S_n\}$
8: $\quad\quad U_g \leftarrow U_g \cup \{G_m\}$

Output: The mapping M

Algorithm 1 Span only Greedy Mapping

The algorithm above iteratively pops the highest Dice score in all the remaining mention pairs. Line 5 ensures that one system mention can be only mapped to one gold mention and vice versa.

Similarly, we create a mapping for attribute-based evaluations. We create all attribute combinations and create the mapping as followed:

Input: A list L of scores $Dice(T_G, T_S)$ for all pair of G, S in the document
Input: The set $\mathcal{A}$ indexing the attributes that will be evaluated for all mentions

1: $M \leftarrow \varnothing$; $U_s \leftarrow \varnothing$; $U_g \leftarrow \varnothing$;
2: **while** $L \neq \varnothing$ **do**
3: $\quad G_m, S_n \leftarrow \arg\max_{(G,S)\in L} Dice(T_G, T_S)$
4: $\quad L \leftarrow L - \{Dice(T_{G_m}, T_{S_n})\}$
5: $\quad$ **if** $S_n \notin U_s$ **and** $G_m \notin U_g$ **and** $Dice(T_{G_m}, T_{S_n}) > 0$ **then**
6: $\quad\quad$ **if** $\mathcal{A}_{S_n} = \mathcal{A}_{G_m}$ **then**
7: $\quad\quad\quad M_{G_m} \leftarrow (S_n, Dice(T_{G_m}, T_{S_n}))$
8: $\quad\quad\quad U_s \leftarrow U_s \cup \{S_n\}$
9: $\quad\quad\quad U_g \leftarrow U_g \cup \{G_m\}$

Output: The mapping M

Algorithm 2 Attribute Augmented Greedy Mapping

Note that "it" and "that" and pronouns including {his, ours, mine, yours, ours, they} are not included in the invisibles list because they can occasionally be resolved as nominal event nuggets.

The attribute-based mapping algorithm add an additional check in line 6 to greedily search for the current best Gold System mapping that also matches their attributes.

3.1.2 Scoring mention detection

To score mention detection, a mention-based F1 score is computed in the following way:

1. For each gold standard mention G, recall that G can be mapped to multiple system mentions, we only choose one system mention S that maximize $Dice\ (G, S)$, and denote $TP_i = \max Dice\ (G, S)$.
2. True Positive = $\sum_i TP_i$
3. Precision = True Positive / #System Mention
4. Recall = True Positive / #Gold Mention
5. F1 = H (Precision, Recall), where H is the harmonic average function

Given a mapping from Algorithm 1, we can simply get the true positive by aggregate the matching score of each gold standard mention:

Input: The set of gold standard mentions $\mathcal{G}$;
Input: The mapping M indexed by gold standard mentions;
1: $TP \leftarrow 0$
2: **for** $G \in \mathcal{G}$ **do**
3: $\quad (S, Dice) \leftarrow M_G$
4: $\quad TP \leftarrow TP + Dice$
Output: TP

We can then simply compute F1 score with the following:

$$P = \frac{TP}{N_s}\ ;\ R = \frac{TP}{N_G}; F1 = \frac{2PR}{(P+R)}$$

We will then iterate through all possible mappings, taking the highest score as the score for the system.

3.1.3 Scoring realis status and mention type detection

To score realis status and mention type detection, we augment the Span-based F-1 score slightly. The only difference is that we will use the mapping computed using the attributes under evaluation. For example, to evaluate "Realis" and "Type" jointly, we compute the mapping using Algorithm 2, where Input Set **A** will be the set that indexes both "Realis" and "Type" for all mentions.

We can then calculate the attribute augmented TP and F1 the same as above. Our evaluation script will produce scores for all possible attribute combination. In current case, these combinations will be Realis only, Type only and Realis &Type.

3.2 Summarization score

After all documents are scored, we also report scores that give a summary of performance over the whole corpus by taking the average across documents. We use the standard Micro and Macro average definition. Given that the number of mentions in a document is usually small, Macro average scores are subject to high variance. We will use the Micro Average scores for final ranking of systems.

3.2.1 Macro Average Scores (numerical average over the document scores):

Precision_macro = sum of all Precision / #document
Recall_macro = sum of all Recall / #document
F1_macro = 2* Precision_macro * Recall_macro / (Precision_macro + Recall_macro)

3.2.2 Micro Average Scores (sum of the individual true positives, false positives, and false negatives of each mention to calculate the overall F-Score)

Precision_micro = (sum of TP on all docs)/ (total number of system mention in all docs)
Recall_micro = (sum of TP on all docs) / (total number of gold standard mention in all docs)
F1_micro = 2* Precision_ micro * Recall_ micro / (Precision_ micro + Recall_ micro)

3.2.3 Note on "invisible words":

Consider the maximum extent of an event nugget, but don't worry about determiners (they are invisible)

- [takes a shower] ==> it is okay for annotators to include "a" in their annotation; we ignore "a" for evaluation
- [make a quick decision] ==> it is okay for annotators to annotate the whole phrase; we ignore "a" and include "quick" in the evaluation

4 Event Coreference Scoring

Our evaluation script will simply call the Reference Implementation of Coreference Scoring algorithms used in CoNLL shared tasks. The script will convert both gold standard and system results to the required format of the CoNLL scorer. We will use the latest version of the scorer[4]. At the time of this writing, the scorer version is v8.01.

4.1 Mapping to CoNLL conversion

The CoNLL conversion will read the mapping between gold and system to produce coreference scores. We take the mention type augmented mapping, which ensures each mapped pair share the same mention type.

We then create a CoNLL mapping from the mention type mapping. We only create a mapping in CoNLL format when the Dice score for the pair of gold and system mention reach a predefined threshold. Following the convention in previous task, we set this threshold to be 1, which means that the mapping can only be valid for exact match cases. However, the scorer can be configured to test the impact of different thresholds.

4.2 Final Score

Systems will be ranked using the unweighted average of the following 4 metrics produced by CoNLL scorer:

1. MUC
2. BCUBED
3. CEAFE (entity based CEAF)
4. BLANC

Note that following the CoNLL evaluation convention, we choose to use CEAFE, and do not include CEAFM (mention based CEAF) in our final scores.

4.3 Problem of Double Annotation

In the current annotation guideline, it is allowed to have double annotations (e.g. the exact span is marked with multiple different event mention triggers). The CoNLL scorer cannot handle such case properly because it distinguishes mentions solely based on the mention span.

Let's consider the following example:

```
#BeginOfDocument sample
gold    sample    E1    t1    murder    Life_Die          Actual
gold    sample    E2    t1    murder    Conflict_Attack   Actual
gold    sample    E3    t2    kill      Conflict_Attack   Actual
@Coreference      R1    E2,E3
#EndOfDocument
```

[4] http://conll.github.io/reference-coreference-scorers/

```
#BeginOfDocument sample
sys     sample      E1      t1      murder      Conflict_Attack      Actual
sys     sample      E2      t2      kill    Conflict_Attack      Actual
@Coreference        R1      E1,E2
#EndOfDocument
```

The gold standard is annotated with 3 event mentions, E1 and E2 shares the same span (t1). The system response only detects 2 mentions, missing one of the double annotated "murder".

When scoring for coreference, the CoNLL scorer will first align the mentions from gold and system with the exact span. System E1 can be aligned to gold E1 or E2, and the resulting coreference score will be different. This is an inherited ambiguity in span representation.

To find the best possible CoNLL score, one need to enumerate all possible alignments and score them all. We find this computational infeasible because exhaustive enumeration is exponential to the number of such cases (a document might contain 50 such cases). Hence we adopt the current greedy method to approximate it.

4.4 Coreference Cluster Validation

We introduced several pre-evaluation checks. These checking are built in the validator and scorer provided. A system response has to pass all the checks in order to get a score; otherwise, the scorer will refuse to do the scoring.

4.4.1 With-in clustering duplication check

This checks that no mentions in the same coreference chain can have the same span. Although two mentions can share the same span, they cannot refer to the same event, otherwise they should be the same mention.

4.4.2 Cluster transitive closure check

The transitive closure of coreference should be solved before submission: if multiple clusters can be merged as a bigger cluster, the submitted response should have written them all in one line. For example:

```
@Coreference        R1      E1,E2
@Coreference        R1      E2,E3
```

is not allowed and should be submitted as:

```
@Coreference        R1      E1,E2,E3
```

Appendix: Tokenization and Event Nugget Boundary Validation

This appendix describes our tokenizer shown in Figure 1. As converted files are provided, participants probably do not need to run this tool by themselves. However, we include the tool description here for the sake of completeness.

This tool is required for the following two purposes:

1. To generate token mapping files.
2. To make sure that boundaries of all event nuggets align with token boundaries.

The first purpose is relatively obvious, and we explain token mapping files below. The second one is important because our evaluation is based on tokens; if the token boundary validation is not guaranteed, one may have event nuggets that cannot be scored by any tokens predicted by a system.

The input and output of our tokenizer are defined as follows.

Input of the tokenizer:

1. Original text files
2. Gold standard annotation files in the brat standoff format[5].

Output of the tokenizer: Token mapping files (.tab)

The token mapping file is a tab-separated file with each line containing the following fields: <token ID><TAB><token string><TAB><begin offset><TAB><end offset>. The first character of the document has an offset of 0. The <begin offset> is inclusive and the end offset can be calculated by <begin offset> + length of span (which make is exclusive).

Below is an example of the token mapping file. Special characters such as "tab" or "new line" are replaced into space for easy manipulation.

```
48    It    386    388
49    has    389    392
50    been    393    397
51    suggested    398    407
52    that    408    412
```

Our tokenizer implementation is based on the tokenizer in the Stanford CoreNLP tool[6]. The software is implemented in Java, and its requirements are as follows:

1. Java 1.8
2. The same number of text files and brat annotation files (*.ann) with the same file base name

You can run the tokenizer by the following command.

[5] http://brat.nlplab.org/standoff.html

[6] http://nlp.stanford.edu/software/corenlp.shtml

If you run the tokenizer without any options, you should see the usage of the software as follows.

```
usage: java TokenFileMaker -a <annotation> -e <extension> [-h]
       -o <output> [-s <separator>] -t <text>
 -a <annotation>   annotation directory
 -e <extension>    text file extension
 -h                print this message
 -o <output>       output directory
 -s <separator>    separator chars for tokenization
 -t <text>         text directory
```

As seen in the usage, the tokenizer takes a text file directory path and an annotation file directory path as input, instead of individual text files and annotation files. The tokenizer outputs the same number of output files as that of input files in the text (annotation) file directory. You can run the tokenizer by specifying options as follows, for example.

```
$ java TokenFileMaker -t ./data/input -a ./data/input -e txt -o
./data/output
```

If the software runs successfully, you might get the following log message without any warning messages.

```
$ java TokenFileMaker -t ./data/input -a ./data/input -e txt -o
./data/output
[INFO] Successfully completed.
```

However, if you get warning messages such as the following, this means that boundary mismatch happened.

```
$ java TokenFileMaker -t ./data/input -a ./data/input -e txt -o
./data/output –s -
[WARN] Boundary mismatch found in d21dc2cb6e6435da7f9d9b0e5759e214: Token
[1716,1735] [buffet/music/buying] vs. EventMentionSpan [1729,1735] [buying]
[WARN] Boundary mismatch found in d21dc2cb6e6435da7f9d9b0e5759e214: Token
[1747,1754] [in/hire] vs. EventMentionSpan [1750,1754] [hire]
[INFO] Successfully completed.
```

This log message shows two boundary mismatches between a system token and an event mention span in the document `d21dc2cb6e6435da7f9d9b0e5759e214`. The first one says that the tokenizer has found a token “buffet/music/buying” whereas a human annotator annotated “buying” as an event mention. In such cases, the software automatically splits a token “buffet/music/buying” into new tokens “buffet/music/” and “buying”.

The final note on the tokenization is a command option to set the additional separator characters for tokenization. The idea of the additional separator characters is to let users control a more fine-grained level of tokenization beyond the Stanford tokenization. Our tokenizer is exactly the same as the Stanford one by default. We observe that sometimes we might want to split the Stanford tokens further (e.g., “buffet/music/buying” into “buffet”, “music” and “buying”). To make

evaluation more flexible, we provide an additional command option `-s`, which defines a set of additional separators for splitting tokens on top of the Stanford tokenizer. For instance, if you use the option in the command above, you will get the following result:

```
$ java TokenFileMaker -t ./data/input -a ./data/input -e txt -o
./data/output –s /
[INFO] Successfully completed.
```

This means that no boundary mismatch is found, since the additional separator character / enables the boundaries of all tokens to be aligned with those of all event mention spans. You can set multiple characters as a string with option `-s`, if necessary. By default, three characters are set: "/", "-", and "\".