Common Sense Knowledge Extraction

Ashish Kumar Saurabh Singh Vatsal Mahajan Vivek Singh
1208657108 1209404713 1209373994 1209521349
Akuma119@asu.edu ssingh139@asu.edu vmahaja1@asu.edu vsingh22@asu.edu

ABSTRACT

This project focuses on extracting common sense knowledge which can be used to generate answers for the Winograd schema challenge. A Winograd Schema consists of a sentence and question pair such that the answer to the question depends on the resolution of a definite pronoun in the sentence. In this project, we extract knowledge base from a corpus which can be used answer such questions. Two approaches have been used to extract knowledge. First approach uses dependency parsed non-ambiguous sentence to create a knowledge base. In second approach, we propose a novel approach to automatically extract knowledge using narrative chains.

1. INTRODUCTION

This report provides a description of our work towards an approach for extracting commonsense knowledge base from a large natural language text corpus. Our primary focus is to extract common sense knowledge based on events (or actions) and their participants; called Event-Based Conditional Commonsense (ECC).

In phase one, we are trying to create a knowledge base which would be storing "Event caused by Property" relationship within a given sentence. We are storing our extracted knowledge in following format:

X.**PROP** = true/false **may cause execution of** A **[ARG*:** X; **ARG*:**Y**]**
where PROP denotes the property causing the action A.ARG:X and ARG:Y denotes the agent and recipient for the action A.For example, we have been consider following sentence:

"Jim yelled at Kevin because Kevin was so upset"

Here Event or Action "yelled" and property is "upset"

With agent as "Jim" and recipient as "Kevin"

The extracted knowledge would be stored in template mentioned above as "Jim.upset = true may cause execution of yelled [ARG0:Jim;ARGM-LOC:Kevin]", This knowledge can be used to resolve ambiguity in a sentence like "Jim yelled at Kevin because he was so upset" by concluding that he refers to Kevin.

In second phase of the project, we are using Narrative chains [3] to extract knowledge base from given sequences of paragraphs in the corpus. Narrative chains are partially ordered sets of events centered around a common protagonist. For example, we are given a sequence of sentences as follows:

"Kevin wanted the ball. Kevin gets the ball as birthday present from John."

Here the common protagonist is Kevin and chain of event is "wanted" and "gets". In above example "wanted" event is causing "gets" event so we would extract the knowledge from sequence of sentences as "Kevin.wanted = true may cause execution of gets[ARG0:Kevin,ARGM:Ball]". Another common protagonist in this sentence is 'ball'.

We extract set of event pairs that share a common protagonist. Then we use labeled Timebank Corpus to create a supervised learning method to classify temporal relation between two verbs as before or after. Using this model, we order the unordered event set into a narration. Rather than creating a chain we simply extract the causal relations to create knowledge templates.

The result of the extraction will be useful for NLU tasks such as hard co-reference resolution, and deep QA require some sort of world knowledge about the problem. Such a common-sense knowledge base will also be useful for Web Semantics, high-level event recognition using reasoning and any system which relies on reasoning over common-sense knowledge base.

2. RELATED WORK

There are various approaches of recognizing causal relations which can be used to extract common sense knowledge. One approach of recognizing these causal relation is done using framenets[1].FrameNet is a manually constructed database based on Frame Semantics. It models the semantic argument structure of predicates in terms of prototypical situations called *frames*. This approach utilizes FrameNet's annotated sentences and relations between frames to extract both the entailment relations and their argument mappings.

The one we are using is closely related to event based commonsense knowledge extraction in ""Automatic Extraction of Events-Based Conditional Commonsense Knowledge" [2].It takes OANC corpus and performs semantic parsing on sentences to extract entities, events and their causal relations.

Then uses Answer set programming to represent common sense knowledge.

3. APPROACH - PHASE 1

3.1 Explanation

Our system relies on the tagging created by Stanford POS tagger. The foremost step was to extract sentences from the OANC corpus. We focused on the written corpuses for extracting sentences. After extracting a sentence, it was tagged by the Stanford POS tagger and checked if it satisfied the requirements for our knowledge base, we are looking for sentences with an event. This sentence was then parsed using the Stanford parser to generate a parse tree. This parse tree is used to find semantic causal relations and create an entry in the knowledge base

3.2 Algorithm for extracting required sentences from corpus

Algorithm 1

1. Extract next sentence from the corpus, until sentences exist.
2. Check if sentence contains a connector, else got to step 1.
3. POS tag the sentence.
4. Divide the sentence into two parts:
   a. sentence_1 = part of sentence before connector
   b. sentence_2 = part of sentence after connector
5. Check if sentence_1 is one of the following types:

.* Noun .* Verb .* Noun .*
Or
.* Pronoun .* Verb .* Noun .*
Or
.* Noun .* Verb .* Pronoun .*
Or
.* Noun .* Adjective .* Noun .*
Or
.* Pronoun .* Adjective .* Noun .*
Or
.* Noun .* Adjective .* Pronoun .*

Else go to step 1.

6. Check if sentence_2 is one of the following types:

.* Noun .* Verb .*
Or
.* Verb .* Noun .*
Or
.* Noun .* Adjective .*
Or
.* Adjective .* Noun .*
Or
.* Pronoun .* Verb .*
Or
.* Verb .* Pronoun .*
Or
.* Pronoun .* Adjective .*
Or
.* Adjective .* Pronoun .*

Else go to step 1.

7. Check if the nouns or pronouns refer to the nouns or pronouns in the first sentence. If reference is ambiguous got to step 1.
8. Save the sentence for creating knowledge.

3.3 Algorithm for extracting knowledge

For extracting knowledge, we generated dependency rules using Stanford Parser for the sentence generated in part a. For example, for the sentence, “The sculpture rolled off the shelf because sculpture was not anchored” Stanford parser generates the following dependency relations:

Universal dependencies

det(sculpture-2, The-1)

nsubj(rolled-3, sculpture-2)

root(ROOT-0, rolled-3)

compound:prt(rolled-3, off-4)

det(shelf-6, the-5)

dobj(rolled-3, shelf-6)

mark(anchored-11, because-7)

nsubjpass(anchored-11, sculpture-8)

auxpass(anchored-11, was-9)

neg(anchored-11, not-10)

advcl(rolled-3, anchored-11)

Algorithm for extracting knowledge templates:

Algorithm 2

1. Take input a sentence generated in step a.

2. Generate dependency rules using Stanford parser.

3. Divide the dependency rules into two parts:

dep_1_1 = Dependency rules before connector

dep2_2 = Dependency rules after connector

4. dep_1 is used to get arg1, arg2 and action for the knowledge:

Nsubj, dobj and other relations are used to find arg1, agr2 and action with respective to a verb.

5. dep_2 is used to find the property or the cause of the action and

I. Advcl, advmod, nsubj relations are used to extract the property associated with an action

II. The existence neg relation is relation is used to mark property false otherwise it is marked as true.

6. Fill knowledge template using the information gathered from step 5 and 6

7. Save the Extracted knowledge.

Template Knowledge:

X.PROP = true/false may cause execution of A [ARG*: X; ARG*:Y]

Extracted Knowledge:

sculpture. anchored = false may cause execution of rolled off [sculpture;shelf]

3.4 Flow Diagram

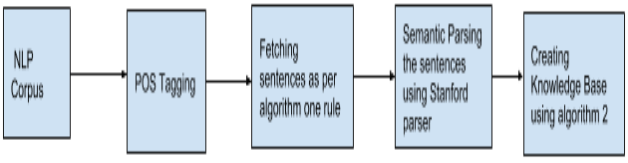


Figure 1 Flow Diagram

3.4 Explanation

Our system relies on the tagging created by Stanford POS tagger. The foremost step was to extract sentences from the OANC corpus. We focused on the written corpuses for extracting sentences. After extracting a sentence, it was tagged by the Stanford POS tagger and checked if it satisfied the requirements for our knowledge base, we are looking for sentences with an event. This sentence was then parsed using the Stanford parser to generate a parse tree. This parse tree is used to find semantic causal relations and create an entry in the knowledge base.

3.5 Execution

3.5.1 Corpus

I. Open American National Corpus (OANC)

The Open American National Corpus (OANC) is a massive electronic collection of American English, including texts of all genres and transcripts of spoken data produced from 1990 onward. All data and annotations are fully open and unrestricted for any use.

II.Wikipedia Corpus

The Wiki corpus is a trilingual corpus (Catalan, Spanish, English) that contains large portions of the Wikipedia (based on a 2006 dump) and has been automatically enriched with linguistic information.

III Clueweb Dataset

The ClueWeb12 dataset was created to support research on information retrieval and related human language technologies. The dataset consists of 733,019,372 English web pages, collected between February 10, 2012 and May 10, 2012.

3.5.2 Knowledge extracted

A sample execution for the extraction of knowledge is as follows:

```
Viveks-MacBook-Air:stanford-parser-full-2015-12-09 viveksingh$ cat result.txt
He was absent from school because of he was sick.
I was absent from school because I had a cold.
We can't trust him because he often tells lies.
She dumped him because he was a jerk.
She asked him to read because she had lost her glasses.
The trophy doesn't fit into the brown suitcase because suitcase is too small.
```

After using the knowledge extraction process as described above we get the following result for the extracted sentences:

```
Viveks-MacBook-Air:stanford-parser-full-2015-12-09 viveksingh$ python knowledgeExtractor.py ou
 he. sick = true may cause execution of absent [ He,school]
 I. cold = true may cause execution of absent [ I,school]
 he. lies = false may cause execution of trust [ We,trust]
 he. jerk = true may cause execution of dumped [ She,dumped]
knowledge not created!!
 suitcase. small = false may cause execution of fit [ trophy,suitcase]
```

4. APPROACH – PHASE 2 (KNOWLEDGE EXTRACTION FROM NARRATIVE CHAINS)

The approach we used for our final submission makes use of Narrative chains to extract casual knowledge, i.e., a knowledge of the form A.*x* => B.*y*, where x and y are events that share a participant and A and B are actors. A narrative event chain is a partially ordered set of events related by a common protagonist. We assume that although a narrative has several participants, there is a central actor who characterizes a narrative chain: the protagonist. For example,

The policeman searched the suspect and then arrested the suspect.

In this example, there are two actors: policeman and suspect, there are two events: search and arrest where search and arrest share the same participant policeman. This sentence can be used to generate a knowledge base which in an informal way can be described as:

suspect.search = true may cause execution of arrest(policeman, suspect)

The system developed by us creates chain of two events with a protagonist and later uses this chain to create knowledge base in the following format:

Learning these prototypical schematic sequences of events is important for rich understanding of text. By learning these narrative chains, we try to generate a commonsense knowledge base.

4.1 Event and their participants

To generate a set of events and their participants, we extract a series of event tuples from a corpus. A tuple is of the form:
<(subject1 event1 object1), (subject2 event2 object2>, where event1 and event2 share a common protagonist and *subjectx, objectx* refer to the participants in an *eventx*. After the completion of this step, a set of these tuples is generated. An important point to consider is that, the events in a tuple may or may not be ordered. The ordering is performed in the next step of temporal ordering discussed in details in the section 4.2.
The corpus used in this project is the English Gigaword sample corpus, LDC Catalog No. LDC2003T05. This corpus contains numerous documents and contains of these documents. The corpus file is an XML file with multiple <DOC> elements. Each <DOC> element represents a document and content of this document is encapsulated inside a <TEXT> element. Each of these text elements have multiple paragraphs contained inside <P> tags. The foremost task was to extract documents and sentences for each document from this XML structure.

Once the sentences are extracted, the next step is to extract a set of event tuples for each document, finding events across sentences in that document. In this step, we find the relationship between different predicates or events in subsequent sentences of a paragraph. In our system, we maintain an event tuple in the following form:
<subject, verb, object, typed_dependency, corefering word>
where verb represents the event and the *typed_dependency* can take two values {*subject, object*}. The typed dependency is a way to represent the propagonist. For two events, the protagonist can be a subject for an event and an object for another.
We are collecting the pair of verbs or events which are connected through the same co-referring entity. This information is used to build a verb/dependency graph between various events and calculate how pairs of events are occurring together. *subject* and *object* are the actors involved in this event. We only extract knowledge where exactly two actors are involved in each event.
Given this graph, verbs can be clustered to create narrative chains with multiple narrative events. Our system creates the verb/dependency graph, but our system doesn't perform clustering to create narrative chains with multiple (more than two) narrative events. The reason being the clustering step was not explained elaborately in their research paper by Chambers and Jurafsky(2012).

For a paragraph, we are using the following steps:

a. Create Dependency Graph and generate POS: We are parsing the sentences using Stanford parser and getting the dependency graph.

Output From Console

```
[main] INFO
edu.stanford.nlp.tagger.maxent.MaxentTagger -
Reading POS tagger model from models\wsj-0-18-
left3words-nodistsim.tagger ... done [0.3 sec].
[main] INFO edu.asu.nlp.fall.CreateDependencyParse
-
[main] INFO edu.asu.nlp.fall.CreateDependencyParse
- Kevin_NNP wanted_VBD the_DT ball_NN ._. He_PRP
gets_VBZ the_DT ball_NN as_IN birthday_NN
present_NN from_IN John_NNP ._.
```

b. Coreference Resolution: We are using the openNLP libraries to find how many entities are there and to get the co-referring entity. After resolving coreference, we maintain a data structure to contain

the noun and the pronouns, i.e., an entity and co-referring entity are stored in an array together.

Output From Console

```
[John ]
[birthday present ]
[the ball ]
[Kevin ][He ]
[the ball ]
```

c. Storing Event-Dependency Pair: In this step, we are using the dependency graph from first step to get events which are related to co-referring entity. Here we are referring "verbs" as "events" in the sentences and for each verb we are storing type of relationship with entities either "*nsubj*" and "*dobj*" which would be used for creating knowledge template later.

d. Extracting Head word: In this step, we are storing the event pairs which relate to co-referring entities in the sentences and noun related to event pairs are stored as head words. For example, in the sentence "John loved Mary till she was honest", "Mary" and "she" are co-referring entities and "Mary" is the head word.

Output From Console

```
[main] INFO edu.asu.nlp.fall.CreateDependencyParse
- 4 wanted nsubj 1 kevin
[main] INFO edu.asu.nlp.fall.CreateDependencyParse
- 3 wanted dobj 1 ball
[main] INFO edu.asu.nlp.fall.CreateDependencyParse
- 4 gets nsubj 1 he
[main] INFO edu.asu.nlp.fall.CreateDependencyParse
- 3 gets dobj 1 ball
[main] INFO edu.asu.nlp.fall.CreateDependencyParse
- 1 gets nmod 1 john
[main] INFO edu.asu.nlp.fall.CreateDependencyParse
- first Node : wanted : object second Node : gets :
object
VerbsDependenciesCount [verb1=VerbDependency
[subject=kevin, verb=want, object=ball,
dependency=object, count=0], verb2=VerbDependency
[subject=kevin, verb=get, object=ball,
dependency=object, count=0],
coreferringEntity=ball, pairCount=0]
[main] INFO edu.asu.nlp.fall.CreateDependencyParse
- first Node : wanted : subject second Node : gets
: subject
VerbsDependenciesCount [verb1=VerbDependency
[subject=kevin, verb=want, object=ball,
dependency=subject, count=0], verb2=VerbDependency
[subject=kevin, verb=get, object=ball,
dependency=subject, count=0],
coreferringEntity=kevin, pairCount=0]
```

After this steps we can a set of pair of event tuples available:
{ {<subject1, verb1, object1, typed_dependency1, corefering word>, <subject2, verb2, object2, typed_dependency2, corefering word>}, … }
For each pair, the propagonist (corefering word) will be the same.

e. Verb/dependency graph: In this step we create a verb/dependency graph from the information collected above which can be used for pointwise mutual information(pmi) calculation. The value of pmi is calculated using the following formulae:

$$pmi(e(w,d),e(v,g)) = log\frac{P(e(w,d),e(v,g))}{P(e(w,d))P(e(v,g))}$$

Where e(w,d) is the verb/dependency pair w and d (hit/subj).

$$P(e(w,d),e(v,g)) = \frac{C(e(w,d),e(v,g))}{\sum_{x,y}\sum_{d,f}C(e(x,d),e(y,f))}$$

where C(e(x,d),e(y,f)) is the number of times the two events e(x,d) and e(y,f) had a co-referring entity with typed dependencies d and f. In the verb/dependency graph, each independent <event, typed dependency> tuple represents a node. Two nodes are connected if they have a common protagonist and the edge cost is the *pmi* score calculated per the above formula. We are not using this graph for clustering as the steps mentioned for clustering in the research paper by Chambers and Jurafsky(2012) was not elaborate enough to program. We leave it as a future work.

f. Current Output

Currently, our system creates a graph which has all the detected verbs as nodes. A verb which has occurred multiple times with different typed dependencies (in our case we have two typed dependencies: Subject and Object). We are not performing any clustering on the graph based on the pmi costs. We are using event pairs with shared referring entity to create our knowledge base. A sample output of the step (d) mentioned above is as follows and gets stored in the *output* folder with the file name *"unordered_event.json"*:

Output from Console

```
{
  "eventChains": [
    {
```

```
    "eventChain": [
      {
        "object": "ball",
        "verb": "throw",
        "subject": "john"
      },
      {
        "object": "william",
        "verb": "hit",
        "subject": "ball"
      }
    ]
  },
  {
    "eventChain": [
      {
        "object": "ball",
        "verb": "want",
        "subject": "kevin"
      },
      {
        "object": "ball",
        "verb": "get",
        "subject": "kevin"
      }
    ]
  },
  {
    "eventChain": [
      {
        "object": "ball",
        "verb": "want",
        "subject": "kevin"
      },
      {
        "object": "ball",
        "verb": "get",
        "subject": "kevin"
      }
    ]
  }
 ]
}
```


More details can be found in the README.MD file of the shared project. The output of this step creates an array of unordered narrative chains. The next section focuses on ordering these narrative chains.

4.2 Temporal Relations

Here we perform temporal classification of verb pairs. The Timebank Corpus labels events and binary relations between events representing temporal order. We used classifiers that follows standard feature-based machine learning approaches as described in (Mani et al., 2006; Chambers et al., 2007) and [8] with training data from Timebank Corpus.

ClearTK(a framework for developing machine learning and natural language processing) was used to get the temporal relations between events. The algorithm is described below.

4.2.1 Algorithm

4.2.1.1 Stage1: Learning Event Attributes

We use learn temporal attributes for events in the NYT Corpus using the labeled Timebank Corpus as training-data. Here we learn the five temporal attributes associated with these events as tagged in the Timebank Corpus. (1) Tense and (2) grammatical aspect are necessary in any approach to temporal ordering as they Timebank Corpus define both temporal location and structure of the event. (3) event class is the type of event.

To learn these attributes, we use the following features as in [2]:

Tense	POS-2-event, POS-1-event, POS-of-event, have word, be word
Aspect	POS-of-event, modal word, be word
Class	Synset

Naive Bayes with Laplace smoothing is used to predict the value all 3 attributes. 3 classifiers were used one for each of the attribute.

4.2.1.2 Stage2: Learning Event-Event Features

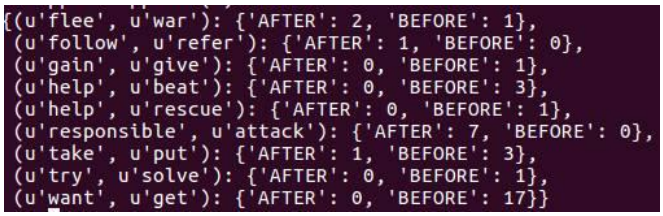
Here we assign temporal relation between events (before/after). Again, we use the TimeBank Corpus as training-data. The features used are as follows [8]

(1) Event Specific: The five temporal attributes from Stage 1 are used for each event in the pair, as well as the event strings, lemmas, and WordNet synsets; (2) POS tags; (3) Event-Event Syntactic Properties: A phrase P is said to dominate another phrase Q if Q is a daughter node of P in the syntactic parse tree. We take dominance as a feature which takes values on/off; (4) Prepositional Phrase: We created feature indicating when an event is part of a prepositional phrase. The feature's values range over 34 English

prepositions; (5) Temporal Discourse: We train two models during learning, one for events in the same sentence, and the other for events crossing sentence boundaries. It essentially splits the data on the same sentence feature.

SVM is used to classify the event pairs form NYT Corpus with labels 'BEFORE' or 'AFTER' denoting temporal relation between them.

The algorithm further counts the number of times two events are classified as before or after. If the number of one kind of the relation (before or after) is more than the other over the complete corpus, then we assign that pair the relation accordingly. The following figure shows a sample of extracted verb pairs with temporal labels.



In every tuple, the first verb is related to second verb using relation(before/after) whichever has higher count.

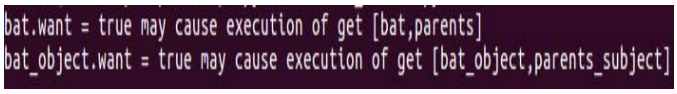
4.3 Creating Knowledge Templates

Now we use the unordered event chains from section 4.1 and verb pair with temporal relations from section 4.2 to create knowledge templates. We create templates of the form;

X.PROP = true/false may cause execution of A [ARG*: X; ARG*: Y]

Intuitively, a statement of the above category means that, the execution of an action A1 may be triggered if property PROP is true or false for an entity X. Here, A1 has X as an argument i.e. X participates in the action A1. Also, we annotate the arguments with their role as subject or object.

The following figure shows a sample of the extracted knowledge.



5. EVALUATION

Here we discuss quantitative evaluation of the knowledge templates extracted in phase-1 and phase-2.

In phase-1 we extracted only 9 instances of knowledge templates from the corpus. The result was less 0.01 % of knowledge extracted per line in the document. So, we manually constructed some sentences and got the template which is shown in above section.

While using narrative chain approach in phase-2 we extract 26% knowledge templates from the unordered event sets extracted from the Corpus. 2100 unordered event sets were extracted from the corpus. The following table shows additional details for phase-2.

Total no. of documents in the corpus	4011
Total no. of lines in the corpus	41771
Avg. no. of lines per document	10.40
Total unordered events	2100
Avg. no. of unordered events per doc	0.52
Total knowledge templates extracted	542

Table: phase-2 quantitative evaluation details.

For qualitative evaluation, we manually filtered the knowledge templates to get 379 out of 542 instances that have relevant Event-Based Conditional Commonsense. After qualitative evaluations, we get extract 18% knowledge templates from 2100 unordered event sets extracted from the Corpus.

Total unordered events	2100
Total knowledge templates extracted.	542
Total knowledge templates extracted after filtering for quality knowledge.	379

Table: phase-2 qualitative evaluation details.

6. RESULTS

We found that the number of quality common sense knowledge extracted depend upon the input data from which knowledge is extracted. The content of quality knowledge extracted is quite less as compared to the corpus since we need to get unambiguous sentences from the corpus which is less than 0.01% in first phase and 18% from 2100 unordered event sets in phase2.

7. FUTURE WORK

The template knowledge we get may not always be correct. Hence, we can weigh these rules based on frequency. Can keep updating these weights on in an online faction. Further the phase2 currently only caters verb-verb causal pairs i.e. including only verb as property/trait. Thus, future work would include even adjectives and nouns as properties/traits.

8. REFERENCES

[1] Aharon, Roni Ben, Idan Szpektor, and Ido Dagan. "Generating entailment rules from framenet." Proceedings of the ACL 2010 Conference Short Papers. Association for Computational Linguistics, 2010.

[2] Arpit Sharma,Chitta Baral Automatic Extraction of Events-Based Conditional Commonsense Knowledge

[3] Nathanael Chambers and Dan Jurafsky. Unsupervised Learning of Narrative Event Chains, http://nlp.stanford.edu/pubs/narrative-chains08.pdf

[4] Nathanael Chambers, Dan Jurafsky. A Database of Narrative Schemas,http://nlp.stanford.edu/pubs/chambers-lrec2010-schemas.pdf

[5] Aharon, Roni Ben, Idan Szpektor, and Ido Dagan. "Generating entailment rules from framenet." Proceedings of the ACL 2010 Conference Short Papers. Association for Computational Linguistics, 2010.

[6] Chambers, Nathanael, and Daniel Jurafsky. "Unsupervised Learning of Narrative Event Chains." ACL. Vol. 94305. 2008.

[7] Arpit Sharma and Chitta Baral. "Automatic Extraction of Events-Based Conditional Commonsense Knowledge." In Proceedings of thirtieth Association for Advancement of Artificial Intelligence workshop. AAAI Press, Workshop on Knowledge Extraction from Text, 2016.

[8] Nathanael Chambers, Shan Wang and Dan Jurafsky, Classifying Temporal Relations Between Events