Naturalie

A Natural Language Inference Pilot Model

Guided by Radhika Mam

Implemented by Ishan and Mukund

AIM

Given 2 pairs of sentences , assuming the first sentence in true , our system needs to output if the sentences were in a ***paraphrase relation or contradictory relation or entailment or neutral to each other.***

Previously...

There have been a ton of implementations of Natural Language Inference (NLI) models, but mostly in English, Chinese etc. or in the domain of Machine Learning. Very few systems actually are primed towards Hindi and are rule based as well.

Most rule based systems have looked at trees, specifically Dependency trees because that bridges Syntax and Semantics in a unique way.

We use some tricks from each domain and then some from Karaka theory.

The Idea

Algorithm Overview

2 s • Serves 4 (types of results)
Results we aim to get -
Neutral, Entailment, Paraphrase, Contradiction

Ingredients & Output

- In - Sentence Pair
- Out - Relation (S1 is assumed True)

Preparation

1. The Sentence Pair is to be passed through a Dependency Parser (end to end).
2. Store the sentences in arrays, and each token is searched for it's synsets and store all the sets separately.
3. Compare Synsets according to and in order of Karaka relations and grade as per scheme

In Detail

Order of Traversing the Dependency Tree and assigning Weights:

- **Main (generally Main Verb)**
- **k1**
- **k1s**
- **k2**
- **K7p / k7t ...**
- **JJ**
- **PRP**

Steps

1) Preprocessing -
   a) If there are lexical items which are used for negation (nahI), we remove them, but for each removal we "flip" the parity of score.
   b) aur and yA, if present, are to be deleted and kept track of, while the two sentences have to be completed on both sides
   c) Run Dependency Parser on these, and store the outputs
2) Following the order on left-
   a) compare word forms (starts with 'main') -
      i) If the same - the all scores except the neutral get updated by 50% of remaining total, except when it's hE (really common).
      ii) Else if word2 is found in word1's synonyms then *all scores* except the neutral get updated by 50% of remaining total.
      iii) Else if w2 is in w1's antonym list, then flip *parity* of score.
      iv) If W1 is only found in Hyponyms of W1 , increase *entailment* scores should be by 50% of remaining total.
   b) Next is k1, if exists -
      i) Check for deictic terms, relating to the Noun you are looking at, if the word forms are the same. Do it by checking if attached PRPs are the same, if so, then update all values by 50%, otherwise just the neutral by 50%.
      ii) If not the same word forms, check and grade synsets as above.

In Detail

Order of Traversing the Dependency Tree and assigning Weights:

- Main (generally Main Verb)
- k1
- k1s
- k2
- K7p / k7t ...
- JJ
- PRP

1) Preprocessing -
2) Following the order on left-
   a) ...
   b) Next is k1, if exists -
      i) ....
      ii) ....
      iii) If the word forms aren't same and nothing updates at the last step
         (1) Then check all the noun relations (k*) in the other sentence.If word forms match
            (a) If found and has a PSP then
               (i) if the PSPs are same and the verbs are same - then update paraphrase by 30%
               (ii) Else just neutral by 30% (werA Gar vs merA Gar)
            (b) Else just neutral by 30% (never found)
   c) Keep following the order and do what was done for k1.
   d) Now for JJ and leftover PRPs do the same thing, but here, just the word forms and the synsets.

Steps

More about the Scoring after Results

Align a node in sentence 1 to the node in the sentence 2 that has both the same Karaka relation and either words are identical, or belong to the same synset in WordNet.

Next slide shows a snippet -

Naturalie's Implementation

Microsoft's Implementation

Part 1 • *Exact and Synonym match*

Align a node in the first tree to any node in the second tree that has both the same part of speech and either words are identical, or belong to the same synset in WordNet.

```
def syn_karaka(arr, rel):  # helper in aligning the sentence structure
   if (word1 == word2 and k1Found == 1 and k1Found2 == 1):
...
   elif (word1 != word2 and k1Found == 1 and k1Found2 == 1):
...
       # compare synsets of the same rel word forms to check if they are related
       arr = syn_compare(word1, word2, arr)
def syn_compare(word1, word2, arr):
   #arr [paraphrase,entailment,neutral]
...
   # first check if wordforms match
   elif word1 == word2 and word1 != "है" and word2 != "है":
...
   elif word2 in syn1["Syn"]:  # check if words are synonyms
...
   elif word2 in syn1["Ant"]:  # check if words are antonyms
...
   elif word2 in syn1["Hyp"]:  # check if words are hyponymous
...
   return arr
```

We completely avoid finding relations by using Karakas. And we look for Antonymy at lexical level within the synsets itself.

As above slide showed

Naturalie's Implementation

Microsoft's Implementation

Part 2 • *Antonym Match*

If two aligned noun nodes (h1,t1) are both subjects or both objects of verb nodes (h0,t0) in their respective sentences, then they check for an antonym match. They construct the set of verb antonyms using WordNet.

Vegetables and Spices needed -

1) Dataset of our own (60 sentences)

   (will peek after presentation, more on this after scoring slide)

2) ISC NLP ‘s Parser
3) IIT Bombay’s Indowordnet database, shrunk down for the pilot.

Problems and why just a Pilot -

1) ISC NLP ‘s Parser - or any other existing Hindi Dependency parser, has its own limits, thus while making the dataset, we kept in mind what kinds of Karaka relations or word senses the Parser couldn’t parse and id.
2) IIT Bombay’s Indowordnet - had a lot of sense relations missing, like asII and nakII weren’t in the antonymy lists of each other.

Future

- Tree editing
- More Nouns and Verbs, apart from Acronyms, Numbers etc.
- A more Robust Scoring, which tested on better and larger data
- Working on not just sentences but ones with subordinating clauses, complex constructions etc.

Future and Result

Results are on the next slide and there are examples as well as non examples.

In Future, we would definitely like to be able to work with chunks rather than POS tags. And with a more regular parser. This would help us look at the problem from a more “tree” perspective, the problem we originally wanted to solve.

```
sent1 = "मेरा घर छोटा है |"
sent2 = "मेरा घर छोटा नही है |"
```

```
[-8.775, 7.5, 0]
Contradiction
```

```
sent1 = "राम छोटा और शक्तिशाली है |"
sent2 = "राम शक्तिशाली है |"
```

```
Entailment
```

```
sent1 = "पानी ठंडा है |"
sent2 = "पानी गरम नही है |"
```

```
[7.5, 0, 6.5]
Paraphrase
```

```
sent1 = "राम ने श्याम को देखा |"
sent2 = "श्याम ने राम को देखा |"
```

```
[2.0, 2.0, 5.1]
Neutral
```

```
sent1 = "कमरे मे गाना चाहिए |"
sent2 = "मेरा घर कमजोर है |"
```

```
[0, 0, 0]
Neutral
```

Scoring and why it works

Why it works?

- As you go down the tree levels, you are moving farther away from the root, thus lesser and lesser important.
- Also, if we went linearly, we couldn't know if there wasn't a k1. This, method ensures that if you don't find k1, but k7p is at the top, then k7p gets more weighed.

Scoring

- As you go down the tree levels, the priority / weights changed, decrease from 50% to 30% to 20%.
- Antonymy and nahl multiply -1 each time we see them
- Entailment specially increases if hyponyms are encountered.
- For aur , if the second sentence is paraphrase with either of the first's parts, then its eventually an entailment
- For yA, if either is Paraphrase, then the result is Neutral, otherwise Contradiction

Dataset

What we have hand made

- Pairs of sentences , which fit into one of the 4 relations and work well through dependency and have good synsets.
- Made Gold Labels for them
- Marked the non-examples - i.e. example pairs that don't output right.

Naturalie

Thanks!