Computational Linguistics Final Project Report

Ishan Agarwal

25 July 2017

Proposal

We propose to attempt a classification of nouns in WordNet as 'artifacts' in the sense of the term in generative lexicon theory. Furthermore we propose to look at the synset containing 'artifact' in WordNet and all it's hyponyms so as to study the overlap between this set of nouns and the nouns our algorithm classifies as artifacts based on the glosses of their synsets.
Part of the motivation for this is to understand the difference between the concept of an artifact in WordNet and the notion of the same in GL theory. We would further like to develop a satisfactory algorithm that can, from reading glosses, pick out the nouns that fit the GL notion of being artifacts. Note that for the moment we are avoiding the problem of polysemy considering the fact that previous work, for example CoreLex, should be able to resolve this.
Furthermore we would like to not only pick out the artifacts from the words in Wordnets list of noun sets but also extract their telic roles from the provided glosses. We aim to try out various algorithms to do this and try to figure out which approaches work and which don't work so well and hopefully why. This would eventually hopefully lead to a better theoretical understanding of the problem as well.

Work done so far

As a first step I tried searching through all the synsets of Wordnet and look at their glosses for the word 'used'. As a typifying feature of artifacts is their having a specific purpose, this was chosen as the starting point.
However this leads to the inclusion of certain extraneous synsets for non physical things like one containing say 'atlantic time'. This was easily remedied by restricting our search to hyponyms of the synset containing 'physical object'.
Furthermore as hyponymy is a transitively closed relation I took the closure over hyponyms of the list of synsets I generated. This is important as many synsets deeper down the hierarchy do not have any typical phrase in their glosses but some hypernyms of these synsets do.
We were also able to compare the list we generated to the set of synsets in wordnet consisting of 'artifact' and all it's hyponyms.

This is the python code to generate our artifact list:

```
import nltk
from nltk.corpus import wordnet as wn

stuff=wn.synsets('object')[0]
hypostuff=set([i for i in stuff.closure(lambda s:s.hyponyms())])
hypo=set([])
artifacts=set([])

for synset in list(wn.all_synsets('n')):
        hypo.clear()
        if((synset.definition().count("used")>0)&(synset in hypostuff)):
               artifacts.add(synset)
               hypo=set([i for i in synset.closure(lambda s:s.hyponyms())])
               artifacts=artifacts.union(hypo)

print(artifacts)
```

Here is the code with some trivial modifications to compute precision and recall:

```
import nltk
from nltk.corpus import wordnet as wn

stuff=wn.synsets('object')[0]
hypostuff=set([i for i in stuff.closure(lambda s:s.hyponyms())])
hypo=set([])
artifacts=set([])
wn_artifacts=set([])
overlap=([])
wn_artifact = wn.synsets('artifact')[0]
wn_artifacts = set([i for i in wn_artifact.closure(lambda s:s.hyponyms())])

for synset in list(wn.all_synsets('n')):
        hypo.clear()
        if((synset.definition().count("used")>0)&(synset in hypostuff)):
               artifacts.add(synset)
               hypo=set([i for i in synset.closure(lambda s:s.hyponyms())])
               artifacts=artifacts.union(hypo)

overlap= artifacts.intersection(wn_artifacts)
count_ours=len(artifacts)
count_theirs=len(wn_artifacts)
count_overlap=len(overlap)
```

```
precision=count_overlap/count_ours
recall=count_overlap/count_theirs

print(precision)
print(recall)
```

We first extracted the artifacts from wordnet by simple searching for keywords such as 'used' or 'designed' as described above. We used a few different algorithms to attempt to extract the telic role:

- 1) We simply return the noun verb pair closest to a keyword in the gloss which is after the said keyword. This is based on the heuristic that roles are generally consist of a noun and a verb and if the sentence construction is simple they are likely to be in proximity to a word signalling the presence of a telic role and after it.
  Here is the code:

```
    import nltk
from nltk.corpus import wordnet as wn
stuff=wn.synsets('object')[0]
hypostuff=set([i for i in stuff.closure(lambda s:s.hyponyms())])
hypo=set([])
artifacts=set([])
n=0
v=0

flag=0
length=0
a=list()
b=list()
c=list()
d=list()
glosses=list()
counter=0
stop=5000
for synset in list(wn.all_synsets('n')):
        stop=stop-1
        if stop<0:
                break
        hypo.clear()
        if(((synset.definition().count("used")>0)
        |(synset.definition().count("designed")>0))&(synset in hypostuff)):
               artifacts.add(synset)
               hypo=set([i for i in synset.closure(lambda s:s.hyponyms())])
               artifacts=artifacts.union(hypo)
```

```
        flag=0
        length=0
        counter=0
        n=0
        v=0
        while (counter<1):
             counter=counter+1
             text=synset.definition()
             tokens=nltk.word_tokenize(text)
             this=nltk.pos_tag(tokens)
             length=len(this)-1

             for i in range(0,length):

                if flag>0:
                     if ((this[i][1][0]=='N') & (n==0)):
                            a.append(this[i][0])
                            b.append(synset.lemmas()[0].name())
                            glosses.append(synset.definition())
                            n=1
                     if ((this[i][1][0]=='V') & (v==0)):
                            d.append(this[i][0])
                            v=1
                if this[i][0]=="used":
                        flag=1
             '''if flag!=1:
                 for i in range(0,length):
                     if flag>0:
                         if ((this[i][1][0]=='N') & (n==0)):
                                 a.append(this[i][0])
                                 b.append(synset.lemmas()[0].name())
                                 glosses.append(synset.definition())
                                 n=1
                         if ((this[i][1][0]=='V') & (v==0)):
                                 d.append(this[i][0])
                                 v=1
                     if this[i][0]=="designed":
                          flag=1'''
             if ((v==0) & (flag>0)):
                    d.append("NF")

c=list(zip(b,d,a))
out=list(zip(glosses,c))

target=open("parseout.txt","w")
for item in out:
```

```
        print(item)
        print("\n")
```

- 2) We use a parse tree and simply proceed down the first path from a keyword uptill a certain constant. This modifies the previous idea of spatial proximity to that of proximity in the chain of dependencies.
  Here is the code:

```
    from pycorenlp import StanfordCoreNLP
import json
nlp=StanfordCoreNLP('http://localhost:9000')
glosses=list()
trees=list()
file=open("output.txt","r")
target=open("parse.txt","w")
glosses=file.readlines()
count=1
next1="XXX"
next2="XXX"
next3="XXX"
for stuff in glosses:
     target.write("\n")
     target.write(stuff)
     target.write("\n")
     next1="XXX"
     next2="XXX"
     next3="XXX"


     out = nlp.annotate(stuff, properties={'annotators':
     'tokenize,ssplit,pos,depparse,openie','outputFormat': 'json'})
     for item in out['sentences'][0]['basicDependencies']:
             if ((item['governorGloss']== "used")
             &(item['governor']<item['dependent'])):
                  next1=item['dependentGloss']
     for item in out['sentences'][0]['basicDependencies']:
             if item['governorGloss']== next1:
                  next2=item['dependentGloss']
     for item in out['sentences'][0]['basicDependencies']:
             if item['governorGloss']== next2:
                  next3=item['dependentGloss']

     count=count+1
     if next1!="XXX":
          target.write(next1)
```

```
            target.write("\n")
      if next2!="XXX":
            target.write(next2)
            target.write("\n")
      if next3!="XXX":
            target.write(next3)
            target.write("\n")
```

- 3) We use a dependency parser to generate parse trees for each gloss. Manually we look at many glosses and find typical paths from a keyword to telic roles. We list such paths and our algorithm searches down such paths and returns words along the path that is most consistent with some path in our list.
  Here is the code:

```
    import nltk
from nltk.corpus import wordnet as wn
from pycorenlp import StanfordCoreNLP
import json
nlp=StanfordCoreNLP('http://localhost:9000')
glosses=list()
trees=list()
target=open("parsed_run.txt","w")
p=""
tokens=[["advcl","dobj", "amod"],["advcl","dobj"],["nmod","amod"]]
chk=1
start="used"

st=wn.synsets('object')[0]
hypostuff=set([i for i in st.closure(lambda s:s.hyponyms())])
hypo=set([])
for synset in list(wn.all_synsets('n')):
      p=""
      if(((synset.definition().count("used")>0)
      |(synset.definition().count("designed")>0))&(synset in hypostuff)):
            target.write("\n")
            target.write(synset.lemmas()[0].name())
            target.write("\n")
            target.write("\n")
            target.write(synset.definition())
            target.write("\n")
            start="used"

            stuff= synset.definition()
            out = nlp.annotate(stuff, properties={'annotators':
```

```
'tokenize,ssplit,pos,depparse,openie','outputFormat': 'json'})
for toke in tokens:
    p=""
    for tok in toke:

         for item in out['sentences'][0]['basicDependencies']:
             if ((item['governorGloss']==start)
             & (item['dep']==tok)):
                  if((item['governor']>item['dependent'])
                  &(start== "used")):
                       continue
                  p=p+start+" "
                  start=item['dependentGloss']
         target.write(p)
         target.write("\n")
```

We can of course add more and more lists of paths in our array with further searching which should significantly improve the algorithm as, due to manpower constraints not too much of manual addition has been done.

- 4) Finally we try a combination of the above techniques in which we use the parse tree but proceed along it till a noun -verb pair is reached to avoid not looking down the correct path in case there are multiple paths or the other problem of not extracting words at the right depth of the path.

Results

We were able to generate a list of synsets which are pysical objects and either have the word 'used' in their gloss or are hyponyms of such synsets.
It is unusual that many objects that we considered artifacts were not considered so by wordnet. We obtained a precision of 69.96 percent and recall was only 43.56 percent. While recall was expected to be lower than precision, I expected the precision figure to be significantly higher as we essentially are only counting synsets with the word 'used' in the gloss, which I expected to be an overly stringent criterion that would result in very high precison and quite low recall. I believe that this might have to do with a usage of the word 'used' in the gloss in some manner that I did not forsee and not as denoting that the nouns in the synset were made for some particular use.
I suspect that the word 'used' may be referring to word usage of the nouns in the synset in some cases such as in the kinds of phrases commonly seen in dictionaries like "this word is often used in a negative sense". Here our algorithm will choose the synset where the choice is not justified by the presence of the word use. I plan to manually inspect some of the data to develop heuristics to mitigate this problem.

We use both a strict and relaxed scoring for a sample size of 75 word-gloss pairs. These were extracted from the list oue=r algorithm extraxted from the wordenet set of noun synsets, uniformly randomly. The combined recall and precision score is the harmonic mean.Note that out of 75 extracted words only some were not really artifacts leading to a sample size of 63 actual artifacts. The difference is counted as true negatives in the scoring. The relaxed scheme graded the answer as correct if the output contained either a noun or a verb from the correct telic phrase. The strict scheme awarded 0.4 score for only a matching noun or only a matching verb but 1 if both matched. If there were extraneous nouns or verbs , we counted that as incorrect noun or verb part.

For relaxed grading:

algorithm	precision	recall	combined score
1	82.3	68.3	74.6
2	83.4	69.6	75.9
3	89.2	84.8	86.9
4	89.2	84.8	86.9

For strict grading:

algorithm	precision	recall	combined score
1	72.7	61.9	66.8
2	73.5	60.8	66.5
3	77.6	68.5	72.8
4	80.2	74.1	77.0

Discussion of Results

- Note that the recall is universally found to be worse than the precision. This is a natural consequence of the inability of our algorithms to correct themselves if they find something with a keyword but which is not really an artifact. I believe that this can be solved by a small amount of pattern matching/case checking.

- Another feature to note is that the performance of all our algorithms gets significantly worse under the strict grading. In some sense this is because, as was the initial heuristic part of the correct telic role is in proximity, either physically or in the dependency parse tree to the marker word we are using for identifying the word as an artifact. Basically since the relaxed scheme grades a role as correct even if it only partially finds the role, it naturally results in higher scores.

- Note that under relaxed grading the combined algorithm does no better than algorithm 3. This was expected as it basically refines searching along the parse tree and the relaxed grading does not differentiate between getting a part and the whole of the telic role.

Next Steps

We propose some further related directions of work:

- We can try to better study the overlap between our list and the artifact synset of Wordnet with all its hyponyms, as has been already discussed. This could lead to several interesting insights such as the suitability of wordnet for a GL framework, a better understanding of the notion of an artifact from the point of view of Wordnet and so on.
- We can try to integrate this tool with other tools to develop an ontology keeping in mind GL principles.
- It is likely that our algorithm is not entirely satisfactory at picking out words that native speakers would call artifacts. We can try to improve this crude algorithm and try to see how much more information can be extracted from glosses to reduce both type 1 and 2 errors that we suspect. A first step for this would be to find a satisfactory way to find errors; we would have to set up a standard against which to test the algorithm.
- We could probably significantly improve the algorithms designed to extract telic roles by adding some simple pattern matching for certain phrases that generally lead from our keywords away from the role. Other improvements such as introducing more keywords could also be thought of.
- I believe experimenting with the success rates of these and other algorithms at finding telic roles may be a good way to study the nature of how these are generally encoded in language and this may prove a useful idea to further develop linguistic theory.

Limitations of our approach

There are several severe limitations in the naive approaches we have used though they prove surprisingly effective in getting at least partial information about the telic role. I will try to discuss some of the major ones below:

- We find artifacts by recognizing certain marker words. But this list is small and can clearly lead to both type 1 and 2 errors. Our approach currently has no way of dealing with this. The false positives include phrases like "typically used to denote..." or "generally use derogatorily..." where the keyword used does not (necessarily) denote the existence of a telic role. I believe this can be fixed by adding such phrases and just pattern matching to avoid them. However we would also miss any atifact that does not explicitly have in it's definition that it is used for and designed for something.
- As Marc pointed out, there is no a priori reason to believe that he telic role would be in proximity to the marker that signifies the existence of a

telic role. However all our methods share the idea that we can somehow trace the role out either by just looking near the keyword, whether near in the sentence or near in the dependency parse, or by following some paths originating at the marker word, such as used or designed. I believe that this heuristic has some basis in fact and think that experimental work on a large sample set might confirm it, giving some validity to the way we set about trying to extract telic roles.

References

I studied material from the following to familiarize myself with the Generative Lexicon theory and some basic ideas in computational linguistics:

- The Generative Lexicon ; James Pustejovsky
- The Language of Word Meaning; edited by Pierrette Bouillon and Federica Busa
- Speech and Language Processing ; Daniel Jurafsky and James H. Martin

The following was used as a reference for WordNet:

- WordNet An Electronic Lexical Database ; Christiane Fellbaum

Acknowledgements

I would like to thank

- Nikhil Krishnaswamy who helped me a lot with setting up and learning the new programming environment
- Marc Verhagen who suggested this project idea to me
- Prof. James Pustejovsky for introducing me to his Generative Lexicon theory and giving me an opportunity to work with his group this summer
- Wanda Weinberger, Jessie McShane, Steven Karel and all the people at Brandeis involved with the Brandeis IISc exchange programme for their hospitality and for making this research work ,and indeed this presentation, possible.

Further Material

Related material including some of the python code, raw output txt files, as well as more details in the form of full project reports, can be found at my github: https://github.com/IshanAgarwal/
Computational-Linguistics-Brandeis-Summer-2017 (please look at the Readme for specifications regarding versions of software used)