University of Toronto, Department of Computer Science
CSC 2501/485F—Computational Linguistics, Fall 2015

Assignment 4

Due date: 13h10, Monday 16 November 2015, on CDF.
This assignment is worth 18% of your final grade.

- Please type your answers in no less than 12pt font; diagrams and structures may be drawn with software or neatly by hand.
- Any clarifications to the problems will be posted on the course website and the CDF forums. You will be responsible for taking into account in your solutions any information that is posted there, or discussed in class, so you should check the page regularly between now and the due date.
- This assignment must be done individually; the work that you submit must be your own. You may, and in fact *should*, discuss your work with others in general terms. But you should avoid looking at one another's work or talking about it in detail. If you need assistance, contact the instructor or TA.

Discovering lexical relations

The goal of this assignment is to replicate and extend the system for discovering new lexical relations by Marti Hearst that was described in lecture notes number 7.

Part 1: Finding hypernym relations [40 marks]

With NLTK and the resources described below, and using Hearst's lexicosyntactic patterns, build and evaluate a system for finding suggested hypernym / hyponym relations between words in a corpus of text.

For each suggested relation that you find, maintain a simple measure of *confidence* in its correctness as the count of the number of times you find the same suggestion. (Hearst did not do this.) If the relation is found just once, it is *low confidence*; if it's found twice, call it *medium confidence*, and if it's found at least three times, call it *high confidence*.

A Python module `Asst4` is available on CDF to help you get started.

Corpus: A corpus of about 2.6 million sentences (60 million words) from the 2004 *New York Times*,* tagged with parts of speech,† is available on CDF in `/u/csc2501h/include/a4/nltk/corpora/nyt.zip`. The corpus is about 145 Mb in compressed form, and three times that if uncompressed. But there is no need to uncompress it, as the compressed file can be read directly by NLTK. You can use `Asst4` to access the corpus via NLTK as follows:

```
import sys
sys.path.append('/u/csc2501h/include/a4')
from Asst4 import nyt_big, nyt_mini

for s in nyt_big.tagged_sents():
    # do something with s
```

The Python object `nyt_mini` is a small development corpus that contains 100,000 sentences selected from `nyt_big`. We have biased the selection towards sentences that are likely to match the patterns mentioned in this hand-out. You can use it for developing, tuning, and debugging your code. **For your final submission, you must use** `nyt_big`.

Chunker: The NYT corpus has already been part-of-speech tagged but not partially parsed or chunked. So you will need to use NLTK's regular-expression chunk parser to get the data into a form ready for pattern-matching. The code snippet below creates a regular-expression chunk-parser, then uses it to parse sentences of the NYT corpus. A chunking rule for NPs, `DefaultNpPattern`, is available in `Asst4`:

*The data is from the *New York Times Annotated Corpus*, distributed by the Linguistic Data Consortium and licensed by the University of Toronto. The data is owned by The New York Times Company, who generously make it available for research and education, and is protected by copyright. You must not copy the data, or use it for purposes other than this assignment.

†Thanks to Paul Cook for the tagging and Uli Germann for conversion to the format required by NLTK.

```
DefaultNpPattern = ''.join([r'(<DT|AT>?<RB>?)?',
                            r'<JJ.*|CD.*>*',
                            r'(<JJ.*|CD.*><,>)*',
                            r'(<N.*>)+'])
```

However, you might wish to modify the rule, so to understand this code you should read section 7.2 of Bird *et al*, and you should consult the API documentation to know what to do with the `Tree` objects produced by the chunker. (The code below is available on CDF in `/u/csc2501h/include/a4/A4-sample-code.py`.)

```
import sys                                    # only necessary once
sys.path.append('/u/csc2501h/include/a4') # only necessary once
from Asst4 import nyt_big                     # only necessary once

from nltk.chunk.regexp import *
from Asst4 import DefaultNpPattern
from nltk import Tree

rule = ChunkRule(DefaultNpPattern, 'Chunk NPs')
parser = RegexpChunkParser([rule],
    chunk_node='NP',   # (optional) name for matched substrings
    top_node='S')      # (optional) name for top tree node

taggedText = nyt_big.tagged_sents()
for t in taggedText:
    chunked = parser.parse(t)
    # do something with the chunked sentence
```

Programming hint: The full NYT corpus is so big that you will run into memory problems if you try to generate a list of chunked sentences with code like this:

```
chunked_corpus = [parser.parse(x) for x in nyt_big.tagged_sents()]
```

Instead, process each sentence individually, as shown in the sample code above.

Lexicosyntactic patterns: You should use Hearst's patterns,* which are shown in Table 1, and add your own if you wish. Note that in the patterns, elements in braces are optional — they may be present or not. But elements separated by a '|' in braces require a choice of exactly one — either *and* or *or*, but not both or neither. Adapt the patterns as necessary for your code, and make sure that they are well-documented in your code.

A particular point to note about rule number 3: When you chunk your text, you'll find that *other* is grouped as part of the NP (as it should be). So strictly speaking, you are not looking for *other* NP_j, but for an NP_j that begins with *other*; and then you'll have to remove the *other* when extracting the hypernym. The same may be true, in all rules, for determiners such as *the*, *several*, and *some* (e.g., *university officials, including some deans, several chairs, and the provost*).

*Hearst, Marti. "Automated discovery of WordNet relations." In: Fellbaum, Christiane (editor), *WordNet: An electronic lexical database*. The MIT Press, 1998, pages 131–151.

Table 1. Hearst's patterns for discovering hyponyms in text.

1. **Pattern:** $NP_0\{,\}$ *such as* $NP_1\{,NP_2,\ldots,\{and\,|\,or\}NP_j\}$
   Extracted hyponyms: For each NP_i $(1 \leq i \leq j)$, HYPONYM(NP_i, NP_0)
   Example: Agar is a substance prepared from a mixture of red algae, such as Gelidium, for laboratory or industrial use.
   HYPONYM(*Gelidium, red algae*)

2. **Pattern:** *such* NP_0 *as* $NP_1\{,NP_2,\ldots,\{and\,|\,or\}NP_j\}$
   Extracted hyponyms: For each NP_i $(1 \leq i \leq j)$, HYPONYM(NP_i, NP_0)
   Example: ... works by such authors as Herrick, Goldsmith, and Shakespeare.
   HYPONYM(*Herrick, author*)
   HYPONYM(*Goldsmith, author*)
   HYPONYM(*Shakespeare, author*)

3. **Pattern:** $NP_0\{,NP_1,\ldots,\}$ $\{and \mid or\}$ *other* NP_j
   Extracted hyponyms: For each NP_i $(0 \leq i \leq j-1)$, HYPONYM(NP_i, NP_j)
   Example: Bruises, lacerations, or other injuries ....
   HYPONYM(*bruise, injury*)
   HYPONYM(*lacerations, injury*)
   Example: ... bistros, coffee shops, and other cheap eating places.
   HYPONYM(*bistro, eating place*)
   HYPONYM(*coffee shop, eating place*)

4. **Pattern:** $NP_0\{,\}$ *including* $NP_1\{,NP_2,\ldots,\{and\,|\,or\}NP_j\}$
   Extracted hyponyms: For each NP_i $(1 \leq i \leq j)$, HYPONYM(NP_i, NP_0)
   Example: All common law countries, including Canada and England ....
   HYPONYM(*Canada, common law country*)
   HYPONYM(*England, common law country*)

5. **Pattern:** $NP_0\{,\}$ *especially* $NP_1\{,NP_2,\ldots,\{and\,|\,or\}NP_j\}$
   Extracted hyponyms: For each NP_i $(1 \leq i \leq j)$, HYPONYM(NP_i, NP_0)
   Example: ... most European countries, especially France, England, and Spain.
   HYPONYM(*France, European country*)
   HYPONYM(*England, European country*)
   HYPONYM(*Spain, European country*)

Hint: Along with simple words, WordNet also contains synsets for commonly used phrases, called WordNet compounds. For examples in Table 1, they include *red algae*, *coffee house*, and *eating place*, but not *cheap eating place* or *common law country*. It is up to you to handle these as you see fit.

Evaluation and report: The evaluation of your system will necessarily be only semi-automatic. For each suggested pair your system finds, count how often each of the following holds (count separately for each confidence level):

1. Both words are already in WordNet, and the relation holds between one or more senses of each.
2. Both words are already in WordNet, but the relation is *contradicted* by WordNet for at least one sense of each word. For example, your program might suggest that an animal is a dog when WordNet already says that a dog is an animal.
3. Both words are already in WordNet, but the relation is not present.
4. One or both of the words is missing from WordNet.

Cases 1 and 2 are confirming and disconfirming cases, respectively, and can be counted automatically. Cases 3 and 4 need to be evaluated by a human judge. Choose (at least) 50 examples of each case from your data (or all that you obtain, if fewer than 50), trying to include suggestions from all of your patterns. Score each of these suggestions as either correct, incorrect, or uncertain, according to your best judgement. Try to include both high-confidence and low-confidence examples in your set.

Write a short report on your results, discussing both the successes and shortcomings of of your program and the patterns that you use.

Hint: It will take your program a fair amount of time to work through the entire *New York Times* corpus. Develop and test your program on the small dataset `nyt_mini` before turning it loose on the complete corpus.

Hint: False positives (and false negatives) can provide interesting insights and help you improve your algorithm. For example, instead of just tossing out false positives as wrong during the evaluation, think (and write in your report!) about what went wrong. Is the instance a hopeless case, or could a small (or even big) change in your set-up help get things right? We do not expect you to implement such changes, especially if they are non-trivial. But we do want you to think about them. For analyzing errors, it is often worthwhile to look at things in context. For manual evaluation (and debugging), consider printing out the entire sentence in addition to the candidates to be evaluated, so that you can look at them in context. But for your submission, print out only the suggestions from your program without context.

Part 2: Finding causal relations [20 marks]

Extend your system to look for causal relations such as *rain causes flooding* and *dehydration causes headaches*. This will require a new set of lexicosyntactic patterns (in fact, they won't just be lexicosyntactic any more) and some modifications to your code from part 1. While causal relations can be found with patterns — for example, the pattern "NP *causes* NP" is a pretty good one — many of the indicators are rather more vague or ambiguous than those of hyponymy. One way to make the patterns more precise is to put restrictions on the verbs and NPs in the patterns. In particular, we require that verbs indicate a causation relation and if the head nouns in the NPs are already in WordNet, then they be hyponyms of a particular root synset. Research by Roxana Gîrju* resulted in a set of causal verbs which are shown in Table 2 (available on CDF as `Causal-verbs.txt`) and a set of patterns which are shown in Table 3 in decreasing order of accuracy. Each pattern takes the form "NP_1–VE–NP_2", where *VE* is a *verbal expression* from Table 2 such as *create*, *bring on*, and *give rise to*. The constraints are either word-matches for the verbal expressions (after morphological analysis) or are WordNet hypernym synsets for the NPs; "!" means "not", a comma means "and", and "$*$" means "any synset" or "any verb from Table 2". To be considered an indicator of a causal relation, an NP_1–VE–NP_2 sequence must match one of the rules in the first part of the table, and must *not* match one of the rules in the second part.

For example, the sentence *An inadequate dietary intake leads to immunodeficiency in children* will match rule C7 because *immunodeficiency* matches !ENTITY, !GROUP (not an entity and not a group), and so we record it as a causal relation. However, the sentence *Behind the church is a path that leads to the cemetery* doesn't match because *cemetery* is an ENTITY, and the relation is recognized as not causal.

Important note: Gîrju's patterns were developed for an early version of WordNet, and they do not work on the present version because some of the higher-level synsets that they rely on were re-arranged between versions. So for this part of the assignment you must use WordNet version 1.7, which is available on CDF in `/u/csc2501h/include/a4/nltk/corpora/wordnet`. NLTK will use these files instead of the newer version of WordNet if the NLTK data search path is adjusted accordingly. We've done that in the Python module `Asst4`, so make sure you import WordNet through the module `Asst4`:

```
import sys # necessary only once in your code
sys.path.append('/u/csc2501h/include/a4') # necessary only once
from Asst4 import wn17 as wn
```

Modify your program from part 1 so that it can take causal verbs, WordNet root synsets, and negative patterns into account in the pattern matching, and use these rules to find suggested causal relations in the *New York Times* Corpus. Evaluate your results by choosing 50 suggestions (or all that you obtain, if fewer than 50) and scoring them as correct, incorrect, or uncertain, according to your best judgement. (Don't expect your results to be as good as for hyponymy!) Write a short report on your results, discussing both the successes and shortcomings of your program and Gîrju's patterns.

*Gîrju, Roxana (2003). Automatic detection of causal relations for question answering. *Proceedings of the ACL 2003 Workshop on Multilingual Summarization and Question Answering*, Sapporo, Japan, 76–83. Association for Computational Linguistics.

Table 2. Gîrju's list of verbal expressions.

activate	effect	originate
actuate	effectuate	originate in
arouse	elicit	produce
associate with	entail	provoke
begin	evoke	put forward
bring	fire up	relate to
bring about	generate	result from
bring forth	give birth to	set in motion
bring on	give rise to	set off
call down	implicate in	set up
call forth	induce	spark
cause	kick up	spark off
commence	kindle	start
conduce to	launch	stem from
contribute to	lead off	stimulate
create	lead to	stir up
derive from	lead up	trigger
develop	link to	trigger off
educe	link up	unleash
	make	

Table 3. Gîrju's patterns indicating causal and non-causal relations. The WordNet 1.7 synsets are STATE.n.04, EVENT.n.01, PHENOMENON.n.01, GROUP.n.01, ACT.n.02, ENTITY.n.01, ABSTRACTION.n.06, and POSSESSION.n.02, which are root nodes of different noun hierarchies in WordNet 1.7. A rule matches any NP whose head is a hyponym of the root synset specified in the rule.

	NP_1	**V**	**NP_2**
	Rules indicating causal relations:		
C1	*	cause	*
C2	*	*	PHENOMENON
C3	!ENTITY	associated with	!ABSTRACTION, !GROUP, !POSSESSION
C4	!ENTITY	related to	!ABSTRACTION, !GROUP, !POSSESSION
C5	!ENTITY	*	EVENT
C6	!ABSTRACTION	*	EVENT *or* ACT
C7	*	lead to	!ENTITY, !GROUP
	Rules indicating non-causal relations:		
N1	*	induce	ENTITY *or* ABSTRACTION
N2	*	*	GROUP, !STATE, !EVENT, !ACT
N3	ENTITY	*	!STATE, !EVENT, !PHENOMENON

Hint: As noted in Part 1, along with simple words, WordNet also contains commonly used phrases (e.g., *chain_reaction*, *human_genome_project*). For better results, check the NPs for such compounds before checking for the NP restrictions.

Hint: There are two key differences between this part and part 1. (1) In this part, all the patterns take the form "NP_1–VE–NP_2". (2) And in this part, the verbal expression may be constrained lexically (must match a particular word or string of words), but the NPs may have additional *semantic* constraints: the head noun must be (or must not be) a hyponym of some specified WordNet synsets. (Strictly speaking, of course, when we say that a noun is a hyponym of some synset, we mean that at least one of the noun's synsets is a hyponym of that synset.)

What to submit

What to submit electronically Please submit electronically your code (for parts 1 and 2), your raw output (all suggestions as `out1.txt` and `out2.txt` for parts 1 and 2, respectively), and your reports (including evaluation of the suggestions as `report1.(pdf|doc(x)|rtf)` and `report2.(pdf|doc(x)|rtf)` for parts 1 and 2, respectively). Submit the files using the `submit` command on CDF:

```
% submit -c <course> -a Asst4 <filename-1>...<filename-n>
```

where `<course>` is `csc485h` for undergraduates and `csc2501h` for graduate students, and `<filename-1>` to `<filename-n>` are the *n* files you are submitting. Make sure every file you turn in contains a comment at the top that gives your name, your login ID on CDF, and your student ID number.

Grading scheme

Part 1

Programming, 20 marks:
Correctness 7; Patterns 4; Coverage 6; Structure 3.

Report, 20 marks:
Automatic evaluation 4; Manual evaluation 4; Discussion 12.

Part 2

Programming, 10 marks:
Correctness 4; Patterns 2; Coverage 2; Structure 2.

Report, 10 marks:
Evaluation 4; Discussion 6.