Gabriel Luong
996268275
CSC485 Assignment 4

Part 1.

To find hypernym relations, regular expressions for each Hearst's pattern was used to match possible hypernym relations. Afterwards, the extract hypernym relations are evaluated for 4 different cases with WordNet to determine if the relation is valid. In building the regular expression for extracting the relations, regular expressions to extract key components in the Hearst's pattern were created and used to generate new regular expressions.

These components included:
{, NP_2, ..., {and|or} NP_j}, {,}, NP_0{, NP_1, ...,}, {and|or}

When extracting hyponyms in patterns such as NP_0{, NP_2, ..., {and|or} NP_j}, each hyponym NP is extracted out after capturing the entire pattern.

For NP which might contain more than one N, the individual nouns are extracted, and joined by a _. Examples of this included box_office, political_leaders, etc. This was effective for getting the proper WordNet synset.

On evaluating the correctness of the 4 different cases for the pair the program found, overall, the consensus was that most pairings were correct with a few exceptions. These examples can be examined in the Part 1 Evaluation. However, some examples included HYPONYM(japan, chefs), and HYPONYM(picture_frames, tokyo) for case 1. This seems to be a problem with the relations defined in WordNet.

In case 2, the pairs found were a bit more ambiguous since it was possible that in some examples that the hyponym should be hypernym and vice versa. This was illustrated in the examples HYPONYM(theft, abuse) and HYPONYM(crappies, panfish). In both case 3 and 4, there were a significant number of correct pairs despite the fact that WordNet may not have a relation for the pair. However, there were a number of nonsense pairs that were reported for case 3 and 4 as a result of WordNet not knowing the relations between the words. These included false positive such as HYPONYM(the_professor, students), and HYPONYM(autism, childhood).

One consideration that was not taken into account by the chunk parser was that the NP extracted is coming from a prepositional phrase in NP PP. The chunk parser should be modified to extracted the central NP in NP PP instead of the NP from PP, which may be defined as PP -> P NP. In additional, the NP extracted can be improved by removing quotations. There were also pairs containing numerical NPs such as HYPONYM(dinner, $150), which are incorrect. To mitigate this case, a new rule to Hearst's pattern could be used to discard numerical NPs or perhaps numerical quantities should not be classified as NPs by the POS tagger.

Part 1 Evaluation of Suggestions.

Legend
======
C = Correct
I = Incorrect
U = Uncertain

Case 1

| Correctness(C, I, U) | Low |
|---|---|
| C | HYPONYM(argentina, countries) |
| C | HYPONYM(jewelry, creations) |
| C | HYPONYM(prosecutors, critics) |
| C | HYPONYM(martin_luther_king_jr., leaders) |
| U | HYPONYM(gangs, inmates) |
| I | HYPONYM(grants, distractions) |
| I | HYPONYM(picture_frames, tokyo) |

C HYPONYM(catalogs, promotional_materials)
C HYPONYM(torture, ill-treatment)
I HYPONYM(court, lawyers)
C HYPONYM(deer, mammals)
C HYPONYM(churches, forums)
C HYPONYM(floor_plans, materials)
I HYPONYM(japan, chefs)
C HYPONYM(helmets, equipment)
I HYPONYM(state, political_leaders)
I HYPONYM(box_office, revenues)
C HYPONYM(stalin, leaders)
C HYPONYM(information, data)
C HYPONYM(kidneys, organs)
C HYPONYM(schools, properties)
C HYPONYM(laboratories, equipment)

Correctness(C, I, U) Medium
C HYPONYM(broadcast, communications)
C HYPONYM(executives, employees)
C HYPONYM(heart_disease, diseases)
C HYPONYM(britain, european_countries)
C HYPONYM(materials, items)
C HYPONYM(defectors, informants)

Correctness(C, I, U) High
C HYPONYM(heart_disease, illnesses)
C HYPONYM(blood, bodily_fluids)
C HYPONYM(children, relatives)
C HYPONYM(iraq, hot_spots)
C HYPONYM(heart_disease, ailments)
C HYPONYM(pakistan, countries)
C HYPONYM(birds, creatures)
C HYPONYM(denmark, countries)
C HYPONYM(china, countries)
C HYPONYM(afghanistan, hot_spots)
C HYPONYM(pilots, workers)
C HYPONYM(depression, problems)
C HYPONYM(mechanics, workers)
C HYPONYM(esquire, magazines)
C HYPONYM(president_bush, republicans)
C HYPONYM(corn, crops)
C HYPONYM(documents, items)
C HYPONYM(fire, agencies)
C HYPONYM(www, web_sites)
C HYPONYM(posters, items)
C HYPONYM(china, nations)
C HYPONYM(tomatoes, vegetables)
C HYPONYM(passwords, information)
C HYPONYM(members, terrorists)
C HYPONYM(thailand, countries)

Case 2

Correctness(C, I, U) Low
I HYPONYM(europe, cities)
C HYPONYM(t-shirts, merchandise)
U HYPONYM(theft, abuse)
U HYPONYM(crappies, panfish)
C HYPONYM(sociology, disciplines)
I HYPONYM(chancellor, grapes)
C HYPONYM(confessional, places)
U HYPONYM(prosecutors, officers)
I HYPONYM(debts, uncertainties)
I HYPONYM(summer_camp, children)
C HYPONYM(civil_engineers, contractors)
C HYPONYM(nasa, agencies)

```
C                HYPONYM(tiffany_glass, fields)
C                HYPONYM(pet_food, products)
C                HYPONYM(home_appliances, devices)
C                HYPONYM(dixieland, music)
I                HYPONYM(wife, son)

Correctness(C, I, U)        Medium
C                HYPONYM(asthma, conditions)
C                HYPONYM(computers, materials)
C                HYPONYM(insurance_companies, businesses)
C                HYPONYM(testosterone, sex_hormones)
C                HYPONYM(provence, restaurants)
C                HYPONYM(explosives, contraband)
C                HYPONYM(beijing, cities)
C                HYPONYM(burglary, crimes)
C                HYPONYM(steaks, meats)
C                HYPONYM(televisions, devices)
C                HYPONYM(diabetes, illnesses)
C                HYPONYM(farmers, groups)
C                HYPONYM(heroin, opiates)
C                HYPONYM(clubhouses, areas)
C                HYPONYM(airports, sites)
C                HYPONYM(terrorism, threats)
C                HYPONYM(bacteria, organisms)
C                HYPONYM(scanners, measures)
C                HYPONYM(executives, employees)
C                HYPONYM(wife, relatives)
C                HYPONYM(commuters, travelers)
C                HYPONYM(obstruction, charges)
C                HYPONYM(money, resources)
C                HYPONYM(cleveland, cities)
C                HYPONYM(social_security_numbers, information)
C                HYPONYM(volleyball, sports)
C                HYPONYM(medical_care, services)
C                HYPONYM(gasoline, fuels)

Correctness(C, I, U)         High
C                HYPONYM(language, sound)
C                HYPONYM(pakistan, countries)
C                HYPONYM(denmark, countries)
C                HYPONYM(washington, states)
C                HYPONYM(pennsylvania, states)
C                HYPONYM(los_angeles, cities)
C                HYPONYM(china, countries)
C                HYPONYM(alcohol, drugs)
I                HYPONYM(pension_funds, institutions)
C                HYPONYM(india, asian_countries)
C                HYPONYM(italy, european_nations)
C                HYPONYM(tax_credits, measures)
C                HYPONYM(roses, varieties)
C                HYPONYM(syria, countries)
C                HYPONYM(fraud, wrongdoing)
C                HYPONYM(fraud, crimes)


Case 3

Correctness(C, I, U)        Low
C                HYPONYM(screenplays, works)
I                HYPONYM(illnesses, smoking)
I                HYPONYM(immaturity, factors)
I                HYPONYM(relationships, america)
C                HYPONYM(basketball_games, events)
C                HYPONYM(ingrid_bergman, people)
C                HYPONYM(budgeting, features)
C                HYPONYM(food, deliveries)
I                HYPONYM(texas, tuesday)
C                HYPONYM(word_processing, features)
```

U HYPONYM(victories, davis)
U HYPONYM(michael, campers)
C HYPONYM(stock_options, income)
C HYPONYM(rank, insignia)
C HYPONYM(sewers, infrastructure)
C HYPONYM(interviews, experts)
I HYPONYM(victims, laity)
C HYPONYM(lobbyists, special_interests)
I HYPONYM(russia, discussions)
C HYPONYM(concerts, festivities)
C HYPONYM(malnutrition, symptoms)
I HYPONYM(menstruation, taboo)
C HYPONYM(commentary, extras)
I HYPONYM(death, researchers)
I HYPONYM(ones, publishers)
C HYPONYM(african-americans, ethnic_groups)
I HYPONYM(infectious_diseases, children)
I HYPONYM(failure, international_law)
I HYPONYM(sickness, evils)

Correctness(C, I, U) Medium
I HYPONYM(boyfriends, realities)
I HYPONYM(insurance, state_government)
C HYPONYM(contributor, contributions)
U HYPONYM(red_lights, violations)
I HYPONYM(autism, childhood)
I HYPONYM(hotels, ventures)
I HYPONYM(dogwood, varieties)
C HYPONYM(mental_retardation, neurological_disorders)
C HYPONYM(census, data)
I HYPONYM(home_equity_loans, forms)
C HYPONYM(pharmaceuticals, industries)
C HYPONYM(crime, problems)
I HYPONYM(overtime, expenses)
I HYPONYM(mortgages, forms)
C HYPONYM(e-mail, data)

Correctness(C, I, U) High
I HYPONYM(medicare, expenses)
C HYPONYM(manuscripts, works)
C HYPONYM(photography, items)
C HYPONYM(photographs, memorabilia)
C HYPONYM(labor, costs)
I HYPONYM(lavender, varieties)
C HYPONYM(jewelry, valuables)
I HYPONYM(hijackings, types)

Case 4

Correctness(C, I, U) Low
C HYPONYM(builders, local_architects)
I HYPONYM(command_bus, the_companies)
I HYPONYM(the_professor, students)
I HYPONYM(a_10-footer, about_120_trees)
C HYPONYM(theaters, myriad_businesses)
C HYPONYM(san_diego, some_cities)
C HYPONYM(wimax, systems)
C HYPONYM(the_central_bank, agencies)
C HYPONYM(raymond_v._gilmartin, senior_executives)
I HYPONYM(the_conventions, major_political_events)
I HYPONYM(oklahoma_state, the_crowd)
C HYPONYM(ms._boynton, former_executives)
C HYPONYM(decca, labels)
C HYPONYM(a_former_top_aide, the_governor)
C HYPONYM(a_stress_test, requisite_pre-op_exams)
C HYPONYM(a_massive_black_oak, mature_trees)
C HYPONYM(25_hamilton_terrace, rooming_houses)

U HYPONYM(a_costly_pass-interference_penalty, several_lapses)
C HYPONYM(cash_worth, assets)
C HYPONYM(a_shelter, organizations)
C HYPONYM(iran, certain_countries)
C HYPONYM(accounting_scandals, problems)
C HYPONYM(diabetes, common_health_conditions)
C HYPONYM(mr._clean, household_products)
C HYPONYM(fire, emergency_departments)

Correctness(C, I, U) Medium
C HYPONYM(face_painting, activities)
C HYPONYM(police, safety_workers)
C HYPONYM(the_vietnam_war, issues)
C HYPONYM(connecticut, other_states)
C HYPONYM(f._b._i._agents, investigators)
C HYPONYM(brazil, several_countries)
C HYPONYM(i._b._m., companies)
C HYPONYM(''christmas, movies)
C HYPONYM(us_airways, companies)
C HYPONYM(new_york, some_states)
C HYPONYM(drug_overdoses, factors)

Correctness(C, I, U) High
I HYPONYM(run-offs, several_masterpiece_fragments)
C HYPONYM(worldcom, companies)
C HYPONYM(cellphones, electronic_devices)
C HYPONYM(the_war, issues)
C HYPONYM(children, family_members)
I HYPONYM(chief, beth_israel_hospital)
C HYPONYM(new_york, seven_states)
C HYPONYM(bar_association, many_boards)
I HYPONYM(dinner, $150)
C HYPONYM(the_f._b._i., federal_agencies)
C HYPONYM(the_united_states, countries)
I HYPONYM(eliel, masters)
C HYPONYM(california, many_states)
C HYPONYM(drug_dealing, crimes)
C HYPONYM(tyco, companies)
C HYPONYM(emil_jannings, bright_lights)
C HYPONYM(the_euro, currencies)
C HYPONYM(plastic, inexpensive_materials)
C HYPONYM(sandwiches, elaborate_restaurant_fare)

Part 2.

To find causal relations, a regular expression was used to extract NP1 verb {P} NP2. A similar methodology as part 1 was used to join NP with more than one word. There was great diffculty in matching the verb with the list of given causal verbs in Table 2. For the regular expression used to extract the relations, past and plural tenses were considered for the verb. Despite the large number of relations returned by the program, it was found that all relations returned contains a preposition. Some matches may have been lost as a result of the program not matching for causal verbs without the preposition.

The incorrect suggestions pointed in the evaluation are mainly due to the relations defined in WordNet. For instance in any_records-related-to-a_review, I consider records as an entity whereas WordNet suggests that records and entity are not related.

To improve the algorithm used in part 2, extra care would be focused in matching the verb with a causal verb after matching NP1 verb {P} NP2. This led to an additional error in the provided suggestion where "begin in" was returned, which is not defined in Table 2.

Part 2 Evaluation of Suggestions.

```
Correctness(C, I, U) - NP1 -         Verb - Preposition -                NP2
=============================================================================
C                    others -   associated -  with -        the_beef_industry
C                 any_fraud -      related -    to -   the_parmalat_companies
C    felony_murder_charges -      related -    to -           the_commission
C             four_artists -   associated -  with -              primitivism
C              other_items -      created -    in -                  support
C             other_doubts -       relate -    to -                venezuela
C           economic_fears -      related -    to -                  the_war
C                   people -   associated -  with -       the_tomasso_family
C             a_right_turn -        leads -    to -       the_checkout_lanes
C                 eviction -        leads -    to -            family_stress
C            birth_defects -      related -    to -                the_drug
C               13_charges -      related -    to -     the_sarin_gas_attack
C        an_advisory_group -   associated -  with -     the_national_academy
C             other_issues -   associated -  with -                 cleaning
C                  the_joy -   associated -  with -            the_victories
C                conjuring -        leads -    to -            the_emergence
C                 lawsuits -      related -    to -              the_attacks
C                   claims -      related -    to -        the_rescue_effort
C                    costs -      related -    to -               litigation
C          the_cancer_risk -   associated -  with -                      pcb
C                     bans -      related -    to -      the_mad_cow_problem
C  serious_health_problems -   associated -  with -                 steroids
C               some_costs -   associated -  with -                recycling
C                this_tree -        leads -    to -              the_concept
C                 projects -      related -    to -              faith-based
Cintelligence_shortcomings -      related -    to -             the_iraq_war
C            circumstances -      related -    to -                  the_war
C            social_events -      related -    to -    art_basel_miami_beach
C    a_mythological_figure -   associated -  with -         marital_fidelity
C         visits_locations -   associated -  with -            the_beginning
C                the_damage -       caused -    to -                importers
I          the_discipline -       begins -    in -                 practice
C               the_louvre -        leads -    to -                    clues
C                 injuries -      related -    to -               childbirth
C                the_lyrics -       relate -    to -              kierkegaard
C        offensive_prowess -        leads -    to -   regular-season_success
C            hollow_praise -        leads -    to -       poorer_performance
I              any_records -      related -    to -                   a_review
C                this_talk -        stems -  from -                      envy
C        a_news_conference -      related -    to -                 the_case
C                the_risks -   associated -  with -              the_auction
C             ''each_thing -        leads -    to -            another_thing
C                the_worker -   contribute -    to -               retirement
C                     costs -      related -    to -            1,660_layoffs
C   other_antigun_programs -      related -    to -       safe_neighborhoods
C                     lore -   associated -  with -             fred_astaire
C                  expense -      related -    to - stock-based_compensation
C        security_problems -      related -    to -             a_trust_fund
I               the_louvre -        leads -    to -                    clues
C              two_dangers -   associated -  with -                      use
C      the_e-mail_messages -         lead -    to -             a_phone_call
C                everything -   associated -  with -                a_product
C     executive_enrichment -   associated -  with -               the_grants
I         street_criminals -   associated -  with -         jerard_steuerman
I                 everyone -   associated -  with -           the_university
C                 injuries -      started -    to -                     chip
```

Code for Part 1

```
# a4part1.py
#!/usr/bin/env python
# g1luongg
# 996268275
import sys, re
sys.path.append('/u/csc2501h/include/a4')
```

```
from Asst4 import nyt_big, nyt_mini, DefaultNpPattern
# nyt_big  is the full POS-tagged 2004 NY Times corpus.
# nyt_mini is the first 100K lines of nyt_big.
# Use nyt_big for your final submission. You can use nyt_mini
# for testing and debugging your code during code development.
# DefaultNpPattern is a simple baseline pattern for NP chunking

from Asst4 import wn17 as wn
# Import version 1.7 of WordNet.
# Newer versions will not work for Question 2!

# create a chunk parser with the default pattern for NPs
from nltk.chunk.regexp import *
BaselineNpChunkRule = ChunkRule(DefaultNpPattern,
                                'Default rule for NP chunking')
NpChunker = RegexpChunkParser([BaselineNpChunkRule],
                              chunk_node='NP',top_node='S')

from HeartPattern import patterns
# Import Hearst's patterns


def get_confidence(num):
    if num == 1:
        return "Low Confidence"
    elif num == 2:
        return "Medium Confidence"
    else:
        return "High Confidence"


# Return the NP from the tagged NP
# Example: "(NP Mr./NP Everest/NP)" => "Mr_Everest"
def get_np(tagged_np):
    return "_".join(re.findall('(\S+)/\w+', tagged_np)).lower()


# Return a list of the tagged NPs in the matched group of hyponyms.
# Example:
# "(NP grease/NN)  ,/,  and/CC  (NP insect/NN pesticides/NNS)"
#   => ['(NP grease/NN)', '(NP insect/NN pesticides/NNS)']
def extract_hyponyms(match):
    return re.findall('\(NP\s+[^)]*\)', match, flags=re.IGNORECASE)


def find_hypernym_relations(sents):
    # Dictionary that keeps track of the tuple (hyponym, hypernym) pairs and
    # their occurrence count and tagged sentence
    pairs = {}

    for s in sents:
        try:
            # Get an appropriate string representation of the tree structure
            # of the tagged sentence
            tagged_sent = str(NpChunker.parse(s)).replace('\n', '')
            for pattern in patterns:
                matches = pattern.search(tagged_sent)
                if matches:
                    hypernym = get_np(matches.group('hypernym'))
                    hyponyms = extract_hyponyms(matches.group('hyponym'))

                    for hyponym in hyponyms:
                        hyponym = get_np(hyponym)
                        hyponym_pair = (hyponym, hypernym)
                        pairs.setdefault(hyponym_pair, {
                            'count': 0,
                            'sentence': []
                        })
```

```
                        pairs[hyponym_pair]['count'] += 1
                        if not tagged_sent in pairs[hyponym_pair]['sentence']:
                            pairs[hyponym_pair]['sentence'].append(tagged_sent)
        except:
            continue

    return pairs


# Dictionary for each case that contains the confidence level as a key,
# and a dictionary containing the hyponym and hypernym pair, and tagged
# sentence as the value.
case1 = {"Low Confidence": [], "Medium Confidence": [], "High Confidence": []}
case2 = {"Low Confidence": [], "Medium Confidence": [], "High Confidence": []}
case3 = {"Low Confidence": [], "Medium Confidence": [], "High Confidence": []}
case4 = {"Low Confidence": [], "Medium Confidence": [], "High Confidence": []}


# Evaluate the suggested pair for each of the cases and categorize them according
# to their confidence level
def evaluate_cases(hyp_pairs):
    for (pair, data) in hyp_pairs.items():
        hyponym = pair[0]
        hypernym = pair[1]
        count = data['count']
        sentence = data['sentence']

        if is_case1(hyponym, hypernym):
            case1[get_confidence(count)].append({
                'count': count,
                'sentence': sentence,
                'hyponym': hyponym,
                'hypernym': hypernym
            })
        if is_case2(hyponym, hypernym):
            case2[get_confidence(count)].append({
                'count': count,
                'sentence': sentence,
                'hyponym': hyponym,
                'hypernym': hypernym
            })
        if is_case3(hyponym, hypernym):
            case3[get_confidence(count)].append({
                'count': count,
                'sentence': sentence,
                'hyponym': hyponym,
                'hypernym': hypernym
            })
        if is_case4(hyponym, hypernym):
            case4[get_confidence(count)].append({
                'count': count,
                'sentence': sentence,
                'hyponym': hyponym,
                'hypernym': hypernym
            })


# List that keeps track of pairs of hyponym and hypernym that are related.
# Used as an optimization for is_sense_related()
related = []

# Return True if one or more senses of each word is related between the two
# given words, and false otherwise.
def is_sense_related(hyponym, hypernym):
    if (hyponym, hypernym) in related:
        return True

    hyponym_synsets = wn.synsets(hyponym)
```

```
    hypernym_synsets = wn.synsets(hypernym)
    result = False

    # Check that both words are present in WordNet
    if hyponym_synsets and hypernym_synsets:
        for hypernym_synset in hypernym_synsets:
            for hyponym_synset in hyponym_synsets:
                # Find all synsets that hypernyms of the hyponym synset and
                # the hypernym synset, and check if there is any relations.
                result = len(hyponym_synset.common_hypernyms(hypernym_synset)) > 0
                if result:
                    # Add the pair as a tuple to the list related
                    related.append((hyponym, hypernym))
                    break

    return result


# Return whether or not both words are present in WordNet, and a relation holds
# between the one or more senses of each.
def is_case1(hyponym, hypernym):
    return is_sense_related(hyponym, hypernym)


# Return whether or not the relation is contracted by WordNet for at least one
# sense of each word
def is_case2(hyponym, hypernym):
    return is_case1 and is_sense_related(hypernym, hyponym)


# Return whether or not both words are already in WordNet, and the relation is
# not present
def is_case3(hyponym, hypernym):
    return wn.synsets(hyponym) and wn.synsets(hypernym) and \
        not is_case1(hyponym, hypernym)


# Return whether or not one or both of the words is missing from WordNet.
def is_case4(hyponym, hypernym):
    return not wn.synsets(hyponym) or not wn.synsets(hypernym)


# Print the data in the case dictionaries.
def print_case(case_num, case, print_sentence=False):
    for i in range(1, 4):
        confidence = get_confidence(i)

        if confidence in case:
            data = case[confidence]
            print "Case", case_num, "-", confidence, "(", len(data), ")"
            print "=" * 50
            for d in data:
                if print_sentence:
                    print d['sentence']
                print "HYPONYM(%s, %s)" % (d['hyponym'], d['hypernym'])
        else:
            print "Case", case_num, "-", confidence, "(0)"
        print "\n" * 2


if __name__ == "__main__":
    hyp_pair_dict = find_hypernym_relations(nyt_big.tagged_sents())
    # hyp_pair_dict = find_hypernym_relations(nyt_mini.tagged_sents())
    evaluate_cases(hyp_pair_dict)

    print_case(1, case1)
    print_case(2, case2)
    print_case(3, case3)
```

```
    print_case(4, case4)


# HearstPattern.py
# g1luongg
# 996268275
import re

##### Regular expression of Hearst's patterns for discovering hyponyms

### Regular expression components to Hearst's patterns

# Returns a regular expression to extract the hyponym NP from the given query
def re_hyponym(query):
    res = '(?P<hyponym>' + query + ')\s*'
    return res

# Returns a regular expression to extract the hypernym NP from the given query
def re_hypernym(query):
    res = '(?P<hypernym>' + query + ')\s*'
    return res

# Regular expression to extract NP from the tree
re_np = '\(NP[^)]*\)'

# Regular express to extract the hypernym NP from 'other NP'
re_np_other = '\(NP\s+other/\w+\s+' + re_hypernym('[^)]*') + '\)'

# Regular expression to match and not capture the commas
re_comma = '(?:,/,\s+)'
# Matches optional {,}
re_comma_optional =  re_comma + '?'

# Regular expression to match and not capture {and|or}
re_and_or = '(?:and/\w+|or/\w+)?'

# Regular expression for NP_0{, NP_1, ...,}
re_np_comma = re_np + '\s*((' + re_comma + re_np + '\s+)*)?'

# Regular expression for NP_0{, NP_1, ..., {and|or} NP_j}
re_np_and_or = re_np + '\s*((' + re_comma + re_np + '\s+)*' + \
    re_comma_optional + re_and_or + '\s+' + re_np + ')?'


### Regular expression of Hearst's patterns

# Pattern 1: NP_0{,} such as NP_1{, NP_2, ..., {and|or} NP_j}
# \(NP[^)]*\)\s*(?:,/,\s+)?such/\w+\s+as/\w+\s+\(NP[^)]*\)\s*(((?:,/,\s+)\(NP[^)]*\)\s+)*(?:,/,
\s+)?(?:and/\w+|or/\w+)?\s+\(NP[^)]*\))?\s*
re_hypernym_such_as_hyponym = re_hypernym(re_np) + re_comma_optional + \
    'such/\w+\s+as/\w+\s+' + re_hyponym(re_np_and_or)
pattern1 = re.compile(re_hypernym_such_as_hyponym, flags=re.IGNORECASE)

# Pattern 2: such NP_0 as NP_1{,NP_2,..., {and|or}NP_j}
# such/\w+\s+\(NP[^)]*\)\s*as/\w+\s+\(NP[^)]*\)\s*(((?:,/,\s+)\(NP[^)]*\)\s+)*(?:,/,\s+)?
(?:and/\w+|or/\w+)?\s+\(NP[^)]*\))?\s*
re_such_hypernym_as_hyponym = 'such/\w+\s+' + re_hypernym(re_np) + 'as/\w+\s+' + \
    re_hyponym(re_np_and_or)
pattern2 = re.compile(re_such_hypernym_as_hyponym, flags=re.IGNORECASE)

# Pattern 3: NP_0{, NP_1, ...,} {and|or} other NP_j
# \(NP[^)]*\)\s*(((?:,/,\s+)\(NP[^)]*\)\s+)*)?\s*(?:and/\w+|or/\w+)?\s*\(NP\s+other/\w+\s+[^)]*
\s*\)
re_hyponym_other_hypernym = re_hyponym(re_np_comma) + re_and_or + \
    '\s*' + re_np_other
pattern3 = re.compile(re_hyponym_other_hypernym, flags=re.IGNORECASE)

# Pattern 4: NP_0{,} including NP_1{, NP_2, ..., {and|or} NP_j}
```

```
# \(NP[^)]*\)\s*(?:,/,\s+)?\s*including/\w+\s+\(NP[^)]*\)\s*(((?:,/,\s+)\(NP[^)]*\)\s+)*(?:,/,
\s+)?(?:and/\w+|or/\w+)?\s+\(NP[^)]*\))?\s*
re_hypernym_including_hyponym = re_hypernym(re_np) + re_comma_optional + \
    '\s*including/\w+\s+' + re_hyponym(re_np_and_or)
pattern4 = re.compile(re_hypernym_including_hyponym, flags=re.IGNORECASE)

# Pattern 5: NP_0{,} especially NP_1{, NP_2, ..., {and|or}}
# \(NP[^)]*\)\s*(?:,/,\s+)?\s*especially/\w+\s+\(NP[^)]*\)\s*(((?:,/,\s+)\(NP[^)]*\)\s+)*(?:,/,
\s+)?(?:and/\w+|or/\w+)?\s+\(NP[^)]*\))?\s*
re_hypernym_especially_hyponym = re_hypernym(re_np) + re_comma_optional + \
    '\s*especially/\w+\s+' + re_hyponym(re_np_and_or)
pattern5 = re.compile(re_hypernym_especially_hyponym, flags=re.IGNORECASE)

patterns = [pattern1, pattern2, pattern3, pattern4, pattern5]

if __name__ == "__main__":
    print re_hypernym_such_as_hyponym
    print re_such_hypernym_as_hyponym
    print re_hyponym_other_hypernym
    print re_hypernym_including_hyponym
    print re_hypernym_especially_hyponym
    print
    # Sample test to extract the hyponym and hypernym from an example sentence
    # for pattern 1
    s = "(S  With/IN  current/JJ  manufactured/VBN  (NP drainage/NN systems/NNS)  ,/,  (NP
pollutants/NNS)  such/JJ  as/IN  (NP grease/NN)  ,/,  (NP fertilizers/NNS)  and/CC  (NP insect/
NN pesticides/NNS)  make/VBP  their/PP$  (NP way/NN)  into/IN  (NP streams/NNS)  and/CC  (NP
rivers/NNS)  ./SENT)"
    other = "(NP Mr./NP Everest/NP)  and/CC  (NP other/JJ small-business/NN owners/NNS)"
    matches = pattern1.match(s)
    if matches:
        hypernym = matches.group('hypernym')
        hyponym = matches.group('hyponym')
        matches = re.findall('\(NP\s+[^)]*\)', hyponym, flags=re.IGNORECASE)
        print hypernym
        print matches
        print
    matches = pattern3.match(other)
    if matches:
        hypernym = matches.group('hypernym')
        hyponym = matches.group('hyponym')
        print hyponym
        matches = re.findall('(\S+)/\w+', "(NP Mr/NP Everest/NP)", flags=re.IGNORECASE)
        print hypernym
        print matches
```

Code for Part 2

```
# a4part2.py
#!/usr/bin/env python
# g1luongg
# 996268275
import sys, re
sys.path.append('/u/csc2501h/include/a4')
from Asst4 import nyt_big, nyt_mini, DefaultNpPattern
# nyt_big  is the full POS-tagged 2004 NY Times corpus.
# nyt_mini is the first 100K lines of nyt_big.
# Use nyt_big for your final submission. You can use nyt_mini
# for testing and debugging your code during code development.
# DefaultNpPattern is a simple baseline pattern for NP chunking

from Asst4 import wn17 as wn
# Import version 1.7 of WordNet.
# Newer versions will not work for Question 2!

# create a chunk parser with the default pattern for NPs
from nltk.chunk.regexp import *
```

```
BaselineNpChunkRule = ChunkRule(DefaultNpPattern,
                                'Default rule for NP chunking')
NpChunker = RegexpChunkParser([BaselineNpChunkRule],
                              chunk_node='NP',top_node='S')

from GirjuPattern import pattern
# Import Girju's patterns


# Return a list of the regular expression for the given causal verbs
def get_causal_verbs():
    lines = [line.strip() for line in open('Causal-verbs.txt')]

    for i in range(len(lines)):
        new_re = ""
        words = lines[i].split(" ")

        for word in words:
            # Check if the last word for the line matches an optional preposition.
            # If so, remove it from the newly joined word.
            optional_match = re.match('\(\w+\)', word, flags=re.IGNORECASE)
            if optional_match:
                new_re += "(" + optional_match.group() + ")?\s*"
            else:
                new_re += word + "[ds]?\s*"
        lines[i] = new_re

    return lines


# Return the word from the tagged component
# Example: (NP Mr/NP Everest/NP)"" => "Mr_Everest"
def get_word(tagged_component):
    return "_".join(re.findall('(\S+)/\w+', tagged_component)).lower()


# List of regular expression of the given causal verbs
re_casual_verbs = get_causal_verbs()


# Return True if the tagged verb is in the list of causal verb
def is_causal_verb(tagged_verb):
    # Get the verb from the tagged verb
    verb = re.findall('(\S+)/\w+', tagged_verb)[0]

    for re_verb in re_casual_verbs:
        if re.match(re_verb, verb, re.IGNORECASE):
            return True

    return False


def find_casual_relations(sents):
    # Dictionary that keeps track of the tuple (hyponym, hypernym) pairs and
    # their occurrence count and tagged sentence
    result = []

    for s in sents:
        try:
            # Get an appropriate string representation of the tree structure
            # of the tagged sentence
            tagged_sent = str(NpChunker.parse(s)).replace('\n', '')
            matches = pattern.search(tagged_sent)
            if matches and is_causal_verb(matches.group('verb')):
                np1 = get_word(matches.group('NP1'))
                np2 = get_word(matches.group('NP2'))
                verb = get_word(matches.group('verb'))
                preposition = get_word(matches.group('preposition'))
```

```
                if is_causal(np1, np2, verb, preposition):
                    result.append({
                        'NP1': np1,
                        'NP2': np2,
                        'verb': verb,
                        'preposition': preposition
                    })
        except:
            continue

    return result


# Returns True if the given NP1-verb-NP2 indicates a casual relations, and false
# otherwise.
def is_causal(np1, np2, verb, preposition):
    result = False

    if not is_non_causal_relations(np1, np2, verb, preposition):
        if re.match('^cause[ds]?$', verb, flags=re.IGNORECASE):
            result = True
        elif is_hyponym(np2, 'phenomenon.n.01'):
            result = True
        elif re.match('^associate[ds]?$', verb, flags=re.IGNORECASE) and \
            preposition == 'with' and not is_hyponym(np1, 'entity.n.01') and \
            not is_hyponym(np2, 'abstraction.n.06') and \
            not is_hyponym(np2, 'group.n.01') and \
            not is_hyponym(np2, 'possession.n.02'):
            result = True
        elif re.match('^relate[ds]?$', verb, flags=re.IGNORECASE) and \
            preposition == 'to' and \
            not is_hyponym(np1, 'entity.n.01') and \
            not is_hyponym(np2, 'abstraction.n.06') and \
            not is_hyponym(np2, 'group.n.01') and \
            not is_hyponym(np2, 'possession.n.02'):
            result = True
        elif not is_hyponym(np1, 'entity.n.01') and is_hyponym(np2, 'event.n.01'):
            result = True
        elif not is_hyponym(np1, 'abstraction.n.06') and \
            (is_hyponym(np2, 'event.n.01') or is_hyponym(np2, 'act.n.02')):
            result = True
        elif re.match('^lead[ds]?$', verb, flags=re.IGNORECASE) and \
            preposition == 'to' and \
            not is_hyponym(np2, 'entity.n.01') and \
            not is_hyponym(np2, 'group.n.01'):
            result = True

    return result


# Returns True if the given NP1-verb-NP2 indicates a non-casual relations, and
# false otherwise.
def is_non_causal_relations(np1, np2, verb, preposition):
    result = False

    if re.match('^induce[ds]?$', verb, flags=re.IGNORECASE) and \
        (is_hyponym(np2, 'entity.n.01') or is_hyponym(np2, 'abstraction.n.06')):
        result = True
    elif is_hyponym(np2, 'group.n.01') and not is_hyponym(np2, 'state.n.04') and \
        not is_hyponym(np2, 'event.n.01') and not is_hyponym(np2, 'act.n.02'):
        result = True
    elif is_hyponym(np1, 'entity.n.01') and not is_hyponym(np2, 'state.n.04') and \
        not is_hyponym(np2, 'event.n.01') and not is_hyponym(np2, 'phenomenon.n.01'):
        result = True

    return result
```

```
# List that keeps track of pairs of hyponym and hypernym that are related.
# Used as an optimization for is_sense_related()
related = []


# Returns True if the given word is a hyponym of the root synset specified,
# and false otherwise.
def is_hyponym(word, root):
    if (word, root) in related:
        return True

    word_synsets = wn.synsets(word)
    root = wn.synset(root)
    result = False

    # Check that both words are present in WordNet
    if word_synsets and root:
        for word_synset in word_synsets:
            # Find all synsets that hypernyms of the hyponym synset and
            # the hypernym synset, and check if there is any relations.
            result = len(word_synset.common_hypernyms(root)) > 0
            if result:
                # Add the pair as a tuple to the list related
                related.append((word, root))
                break

    return result


# Print the data in the list of casual relations
def print_result(casual_relations, print_sentence=False):
    print '%24s - %12s - %5s - %24s' % ("NP1", "Verb", "Preposition", "NP2")
    for relation in casual_relations:
        print '%24s - %12s - %5s - %24s' % (relation['NP1'], relation['verb'], \
            relation['preposition'], relation['NP2'])


if __name__ == "__main__":
    casual_relations = find_casual_relations(nyt_big.tagged_sents())
    # casual_relations = find_casual_relations(nyt_mini.tagged_sents())
    print_result(casual_relations)


# GirjuPattern.py
# g1luongg
# 996268275
import re

##### Regular expression of Girju's patterns

### Regular expression components to Girju's patterns

# Regular expression to extract NP
re_np = '\(NP[^)]*\)'

# Regular expression for a verb: <verb>/V<pos>
re_verb = '(?P<verb>\w+/V\w*)\s*'

# Regular expression to extract the preposition {to|with|from|in}
re_to_with_from_in = '(?P<preposition>to/\w+|with/\w+|from/\w+|in/\w+)?\s*'


### Regular expression of Girju's patterns
# \(NP[^)]*\)\s*\w+/V\w*\s*(to/\w+|with/\w+|from/\w+|in/\w+)?\s*\(NP[^)]*\)
re_girju = '(?P<NP1>' + re_np + ')\s*' + re_verb + re_to_with_from_in + \
    '(?P<NP2>' + re_np + ')'
pattern = re.compile(re_girju)
```

```
if __name__ == "__main__":
    print re_girju
```