Ricky Ma 82943424
Bruce Cui 13412151
Late days claimed: 0

Question 1: Similarity Metrics in Ontologies

1. 15 synset pairs from WordNet:

Word 1	Word 2	property
push#v#2	thrust#v#1	very similar
help#v#1	aid#v#2	very similar
hammer#n#2	mallet#n#2	very similar
dinner#n#1	lunch#n#1	very similar
car#n#1	gondola#n#3	less similar
authoritarian#adj#1	democratic#adj#3	less similar
footrace#n#1	play#v#18	less similar
rosebush#n#1	galaxy#n#3	very different
sleep#n#1	fork#n#1	very different
lion#n#1	phone#n#1	very different
tax return#n#1	return#v#1	different POS
feed#n#1	feed#v#2	different POS
note#n#1	observe#v#2	different POS
ascent#n#1	climb#v#1	different POS
holiday#n#2	vacation#v#1	different POS

2. Ranking synset pairs by hand

Word 1	Word 2	similarity
push#v#2	thrust#v#1	1
help#v#1	aid#v#2	2
hammer#n#2	mallet#n#2	3
dinner#n#1	lunch#n#1	4
feed#n#1	feed#v#2	5
holiday#n#2	vacation#v#1	6
ascent#n#1	climb#v#1	7
car#n#1	gondola#n#3	8
note#n#1	observe#v#2	9
popular#adj#1	democratic#adj#3	10
footrace#n#1	play#v#18	11
tax return#n#1	return#v#1	12
sleep#n#1	fork#n#1	13
lion#n#1	phone#n#1	14
rosebush#n#1	galaxy#n#3	15

3. Ranking synset pairs using WordNet similarity measures. The results are compared to our predicted rankings, and are colour-coded by the difference in rankings. For example, green represents rankings off by no more than 2 places, whereas red represents rankings off by more than 4 places. The words "authoritarian#adj#1" and "democratic#adj#3" were not recognized by the WS4J web-app, and JCN and path similarities do not support comparisons between words of different parts-of-speeches. These pairs are left out for brevity.

Word 1	Word 2	lesk (lower = more similar)	manual ranking
footrace#n#1	play#v#18	5	11
note#n#1	observe#v#2	11	9
holiday#n#2	vacation#v#1	16	6
ascent#n#1	climb#v#1	19	7
sleep#n#1	fork#n#1	19	13
help#v#1	aid#v#2	20	2
push#v#2	thrust#v#1	26	1
tax return#n#1	return#v#1	29	12
car#n#1	gondola#n#3	47	8
lion#n#1	phone#n#1	51	14
feed#n#1	feed#v#2	52	5
hammer#n#2	mallet#n#2	63	3
dinner#n#1	lunch#n#1	76	4
rosebush#n#1	galaxy#n#3	77	15

Word 1	Word 2	jcn (higher = more similar)	manual ranking
dinner#n#1	lunch#n#1	0.3131	4
help#v#1	aid#v#2	0.0748	2
lion#n#1	phone#n#1	0.0600	7
push#v#2	thrust#v#1	0.0591	1
sleep#n#1	fork#n#1	0.0528	6
car#n#1	gondola#n#3	0.0000	5
hammer#n#2	mallet#n#2	0.0000	3
rosebush#n#1	galaxy#n#3	0.0000	8

Word 1	Word 2	path (higher = more similar)	manual ranking
dinner#n#1	lunch#n#1	0.3333	4
help#v#1	aid#v#2	0.1667	2
hammer#n#2	mallet#n#2	0.1429	3
car#n#1	gondola#n#3	0.0833	5
push#v#2	thrust#v#1	0.0769	1
rosebush#n#1	galaxy#n#3	0.0667	8
sleep#n#1	fork#n#1	0.0625	6
lion#n#1	phone#n#1	0.0588	7

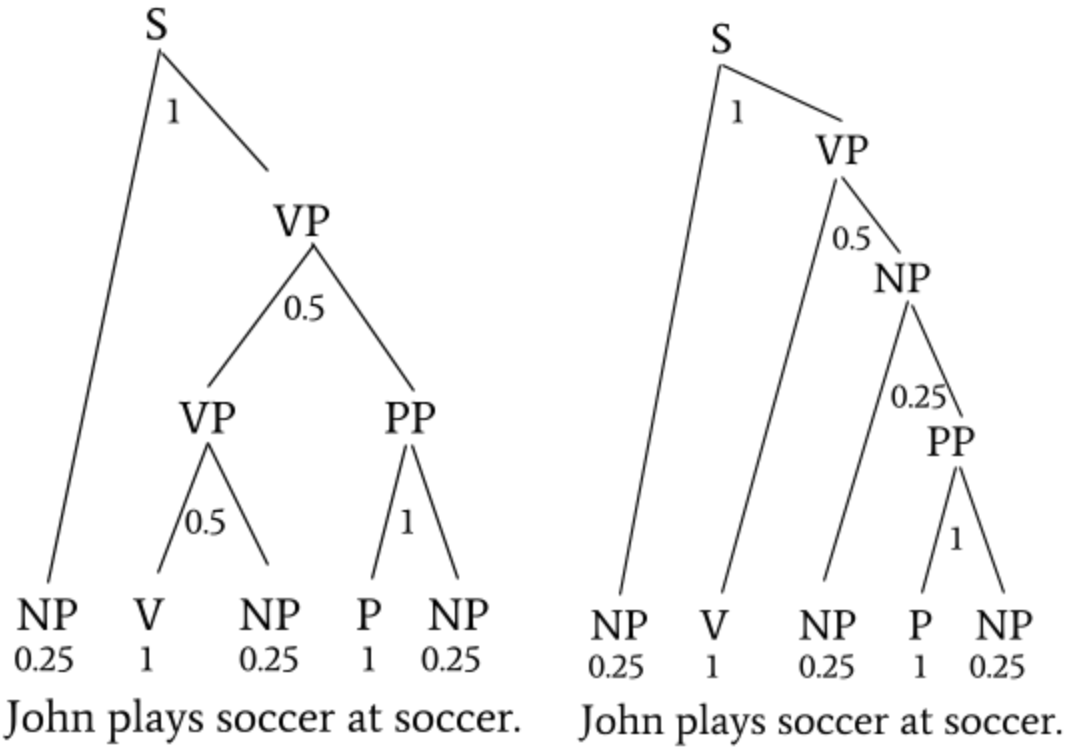
4. The different measures produce different rankings. The measure that is most consistent with our ranking is path similarity. Path and JCN similarity produce similar rankings, while lesk similarity produces wildly different results from what was initially predicted.

   Many of the pairs we predicted to be similar were predicted to be very dissimilar by lesk similarity (i.e. feed#n#1 and feed#v#2, hammer#n#2 and mallet#n#2, dinner#n#1 and lunch#n#1). Additionally, the words push#v#2 and thrust#v#1, which arguably describe the exact same action, is given a surprisingly low similarity measure.

   Some questionable results produced by the JCN similarity measure include the pairs (“lion#n#1”, “phone#n#1”) and (“hammer#n#2”, “mallet#n#2”). The former pair represent very different concepts, while the latter could be synonyms in some contexts.

Question 2: Probabilistic Context-Free Grammar (PCFG)

a. BEST PARSE: (S (NP /John/) (VP (VP (V /plays/) (NP /soccer/)) (PP (P /at/) (NP /soccer/))))



Yes, the algorithm gives the correct parse. Given the grammar rules in pcfg-grammar.pl, the sentence can be interpreted in two ways. The algorithm chooses the parse tree that is more likely.

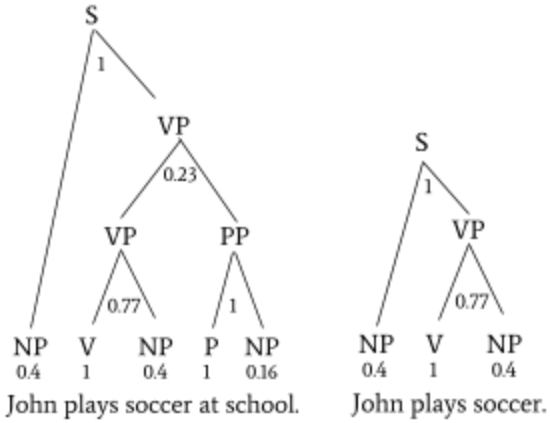
- Left tree (best parse): P(tree) = 0.25*0.25*0.25*0.5*0.5*1*1 = 0.00391
- Right tree (alternate): P(tree) = 0.25*0.25*0.25*0.25*0.5*1*1 = 0.00195

b. Code for automatic grammar generation is attached below.

i. Estimates for grammar probabilities given the “silly corpus”:

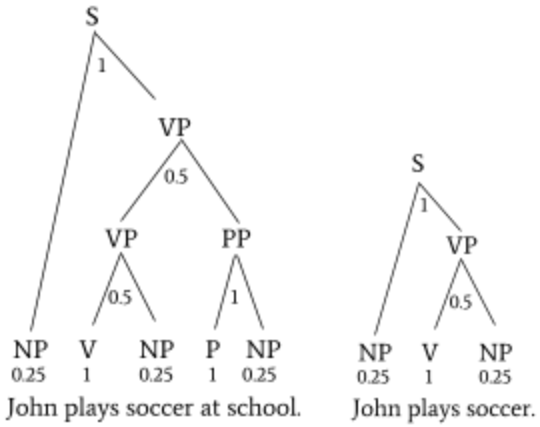
prob{"S -> NP VP"} = 10/10 = 1;
prob{"VP -> V NP"} = 10/13 = 0.7692;
prob{"VP -> VP PP"} = 3/13 = 0.2308;
prob{"PP -> P NP"} = 4/4 = 1;
prob{"NP -> NP PP"} = 1/25 = 0.04;
prob{"NP -> John"} = 10/25 = 0.4;
prob{"NP -> soccer"} = 10/25 = 0.4;
prob{"NP -> school"} = 4/25 = 0.16;
prob{"V -> plays"} = 10/10 = 1;
prob{"P -> at"} = 4/4 = 1;

ii. Probability of most likely parse:



1. *John plays soccer at school.*
   P(tree) = 0.4*1*0.4*1*0.16*0.7692*0.2308*1 = 0.004545
2. *John plays soccer.*
   P(tree) = 0.4*1*0.4*0.7692 = 0.1231

iii. Probability of most likely parse using old grammar:



1. *John plays soccer at school.*
   P(tree) = 0.25*0.25*0.25*0.5*0.5*1*1*1*1*1 = 0.00391
2. *John plays soccer.*
   P(tree) = 0.25*0.25*0.5*1*1 = 0.03125

iv. Using the two different grammars, the probabilities for both sentences are different. This is because with the more intuitive grammar, common words like “John” and “soccer” are more likely. Hence, the probability of a sentence containing the more common words should be more likely. The same applies to phrases. For example, given the “silly corpus”, verb phrases are more likely to be composed of a verb and a noun phrase, than a verb phrase and a prepositional phrase. Hence, the probability of a sentence containing a more common phrase composition should be more likely. The old grammar does not take any of this into account, as all probabilities are uniform.

```
totals = {}
for pos in ['S', 'VP', 'PP', 'NP', 'V', 'P']:
    totals[pos] = corpus.count('(' + pos + ' ')
print(totals)

words = set([x for x in corpus.split('/') if not ' ' in x and not ')' in x])
print(words)

# Get all substrings of the corpus
phrases = [corpus[i: j] for i in range(len(corpus))
           for j in range(i + 1, len(corpus) + 1)]

# Filter results to contain only valid phrases
phrases = [x for x in phrases if
           x.count('(') == x.count(')') and
           '\n' not in x and
           x[0] == '(' and
           x[-1] == ')' and
           x.count('(') % 2 == 1 and
           (x[3] == '(' or x[4] == '(' or x[5] == '(' or x.count('(') == 1)]
print(len(phrases))
```

```
counts = {}
for phrase in phrases:
    # Get main part of speech from phrase
    part_of_speech, *_ = re.findall('\((.*?) ', phrase)

    # Single word
    if phrase.count('(') == 1:
        word = re.search('/(.*)/', phrase).group(1)
        try:
            counts[part_of_speech + '_' + word] += 1
        except KeyError as e:
            counts[part_of_speech + '_' + word] = 1

    # Get count of two-part phrase
    else:
        phrase = phrase[len(part_of_speech) + 2:]
        pos1, *_ = re.findall('\((.*?) ', phrase)
        phrase = phrase[len(pos1) + 2:]
        pos2, *_ = re.findall('\((.*?) ', phrase)
        try:
            counts[part_of_speech + '_' + pos1 + '_' + pos2] += 1
        except KeyError as e:
            counts[part_of_speech + '_' + pos1 + '_' + pos2] = 1
print(counts)

grammar = {}
for pos in counts.keys():
    main_pos = pos.split('_')[0]
    grammar[pos] = counts[pos]/totals[main_pos]
print(grammar)
```

Question 3: IBM Watson and what we covered in 322/422

Based on our research, we believe there are three main components to how Watson works:

1. Understand the problem
2. Search for possible answers
3. Determine the final answer to decide whether to buzz and/or how much money to wager

Understanding the problem

The sources for Watson include a wide range of encyclopedias, dictionaries, thesauri, newswire articles, literary works, and more. Likewise, in order to understand the problem, Watson uses a mixture of techniques, from shallow parses to semantic role labels and more. Parsing the question requires context-free grammar. It is possible that Watson uses a probabilistic CFG with the CKY algorithm to get the most probable parse tree. It then needs to convert the most probable tree into a first order language query, which was not covered in 422..

Relevant quote from the listed link:

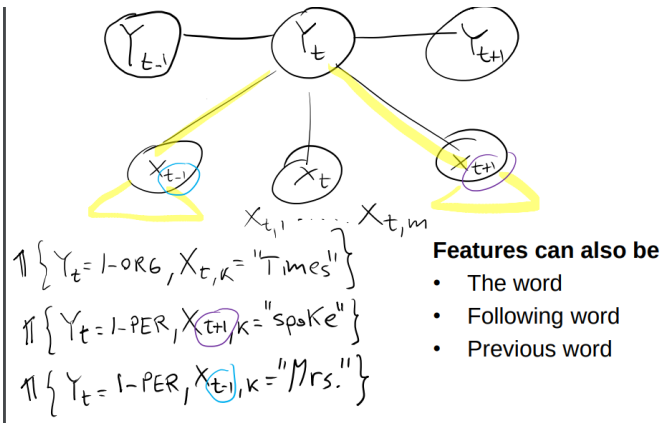
"The DeepQA approach encourages a mixture of experts at this stage, and in the Watson system we produce shallow parses, deep parses (McCord 1990), logical forms, semantic role labels, coreference, relations, named entities, and so on, as well as specific kinds of analysis for question answering."

Searching for possible answers

Next, Watson needs to perform a query on a first-order logic knowledge base. The knowledge base consists of databases like Freebase, taxonomies, and ontologies such as dbPedia, WordNet, and Yago. Assuming Watson processes the question into a queryable format, we need to make sure the FOL KB can answer the question.

Watson then uses syntactic parsing to convert the FOL KB into queryable named entities. There are many techniques to perform named entity recognition and relational extraction, such as conditional random fields or neural networks. The original model, as described by the team at IBM Watson, is "a massively parallel probabilistic evidence-based architecture." However, with the development of more advanced techniques, Watson has shifted to using deep learning and transfer learning, as evidenced here: video.

Lastly, Watson generates several hundred candidate answers. Some may just be a detected relation or a document's title. To produce more candidate answers, Watson may use substring analysis, link analysis, or reverse dictionary lookups. Using a machine learning based approach, it filters the results down to about 100 candidate answers. In CPSC 422, we learned that named entities can be extracted using linear chain CRFs, as discussed in lecture 19.



Relevant quote from listed link:

"Another step in the content-acquisition process is to identify and collect these resources, which include databases, taxonomies, and ontologies, such as dbPedia,7 WordNet (Miller 1995), and the Yago8 ontology. [...] Hypothesis generation takes the results of question analysis and produces candidate answers by searching the system's sources and extracting answer-sized snippets from the search results".

Determining whether to buzz and how much to wager

Many different algorithms are implemented in Watson to calculate the score of an answer. While it was not stated in the article, Waton likely calculates the probability of each answer being correct. To do this, we could build a belief network to calculate P(answer_correct|all_evidence).

Lastly, Jeopardy has a game component of trying to maximize the dollar amount. This can be implemented via reinforcement learning. Specifically, our guess is that:

1. the states can be represented by the number of questions left uncovered on the Jeopardy board, how much money agent currently has, who's turn it is, etc.
2. the actions are to wager or not to wager, and if we wager, the amount of money to wager
3. the reward is the dollar amount

Relevant quote from link:

"These challenges drove the construction of statistical models of players and games, game-theoretic analyses of particular game scenarios and strategies, and the development and application of reinforcement-learning techniques for Watson to learn its strategy for playing Jeopardy. Fortunately, moderate amounts of historical data are available to serve as training data for learning techniques. Even so, it requires extremely careful modeling and game-theoretic evaluation as the game of Jeopardy has incomplete information and uncertainty to model, critical score boundaries to recognize, and savvy, competitive players to account for. It is a game where one faulty strategic choice can lose the entire match."

Sources:

https://ai.stackexchange.com/questions/211/what-are-the-main-ai-technologies-behind-the-watson-platform
https://www.aaai.org/Magazine/Watson/watson.php
https://www.youtube.com/watch?v=r7E1TJ1HtM0&ab_channel=IBMWatson