**Exam 2 Write-up Solutions**

Noah Brackenbury

CS 322, Fall 2015

**1.**

Since part **c)** has a function with cos(16xπ), then it will have a frequency of 8 cycles per second. In order to properly measure a wave, you need to have a sample rate of twice the frequency. Therefore, our sample rate must be at least 16 samples per second.

**a)** cos(8xπ) has maximums at .5, .75, and 1

sin(4xπ) has a maximum at .625

This function together has two maxima roughly at .5 and .75, which repeat. It thus follows that the most contributing frequency would be 4 cycles per second or 4 Hz. I don't know how to represent this in a bin number.

**b)** This function has a period of 1/4, which is again equal to a frequency of 4 cycles per second or 4 Hz.

**c)** This function also experiences two main maxima over this half-second window, so the highest contributing frequency would be 4 Hz. However, it also has a smaller local maximum that would make 8 Hz a less-contributing, but still relevant frequency.

**2. Parsing**

**a)** The Earley algorithm does not need a grammar that is in Chomsky-Normal form, so I would make minor changes to this grammar so it did not have one state leading to three others. This would make making the table easier. I would make VP -> Aux VP NP into VP -> Aux VP, and eliminate VP -> VP NP PP.

**b)**

I can like

S -> .NP VP [0,0] S -> NP.VP [0,1] VP -> .VP NP [2,2]

NP -> .Noun [0,0] NP -> Noun. [0,1] NP -> NP PP. [0,2]

NP -> .NP PP [0,0] NP -> NP.PP [0,1] S -> NP VP. [0,2]

NP -> .NP and NP [0,0] NP -> .Noun [1,1] NP -> Noun. [1,2]

cats and dogs

PP -> Prep.NP [2,3] NP -> NP.and NP [3,4] NP -> NP and.NP [3,5]

VP -> VP.NP [2,3] VP -> VP NP. [2,4] NP -> .Noun [4,5] NP -> Noun. [4,5]

NP -> .NP and NP [3,3] NP -> Noun. [3,4] S -> NP VP. [0,5]

NP -> .Noun [3,3]

**c)**

S{

NP( Noun[ i ] ) VP(

Aux[ can ] VP(

VP( Verb[ like ] ) NP(

NP( Noun[ cats ] ) [and] NP( Noun[ dogs ] ) ) ) }

S{

NP( Noun[ i ] ) VP(

VP( Verb[ can ] ) PP(

Prep[ like ] NP(

NP( Noun[ cats ] ) [and] NP( Noun[ dogs ] ) ) ) }

**3.** In this large treebank data set to train on, you've got a file with a lot of sentences with parse trees. Assuming you can read this file correctly into your model, the first order of business would be to build a model accounting for the simply the word counts of your treebank set. Using <start> and <end> tags for each sentence, I would create a dictionary for word counts and also for bigram counts.

The more difficult part is applying the parse tree to improve the accuracy of your language model. Firstly, if the parse trees are in a Chomsky-Normal form grammar, then each word in the sentence would correspond to a part of speech. These parts of speech could be treated as hidden states and the transition probability and emission probability could be calculated for each, like a POS tagger. This alone would improve the accuracy, but if the parse trees were generated using the CKY-parse algorithm, then each box in the parse grid should also contain the co-ordinates of the two boxes that it draws from. Working our way up the tree in this data set, we could create a dictionary that counts the number of times a nonterminal in the second level of this tree leads to the two terminals that it joins together. When the first terminal is then seen while parsing an input sentence, the probability that the second terminal is next would be weighted higher.

Next, the probabilities within the parse tree can be included in our calculations. If NP leads to Det NP more often than it leads to VP NP, this can be stored in a dictionary. When parsing an input sentence with the given grammar, the probability can be weighted higher if an NP leads to Det NP rather than VP NP.