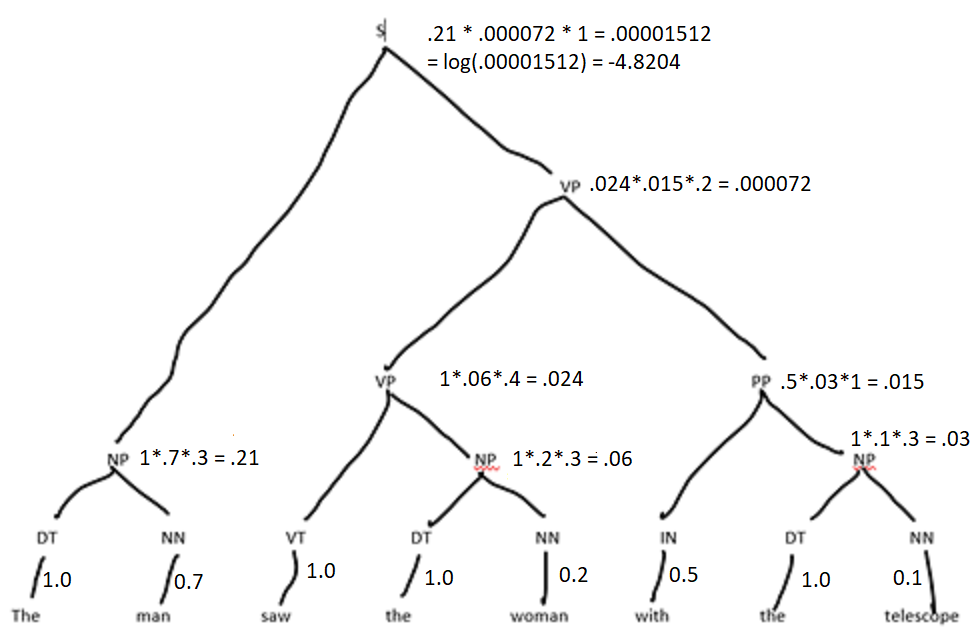
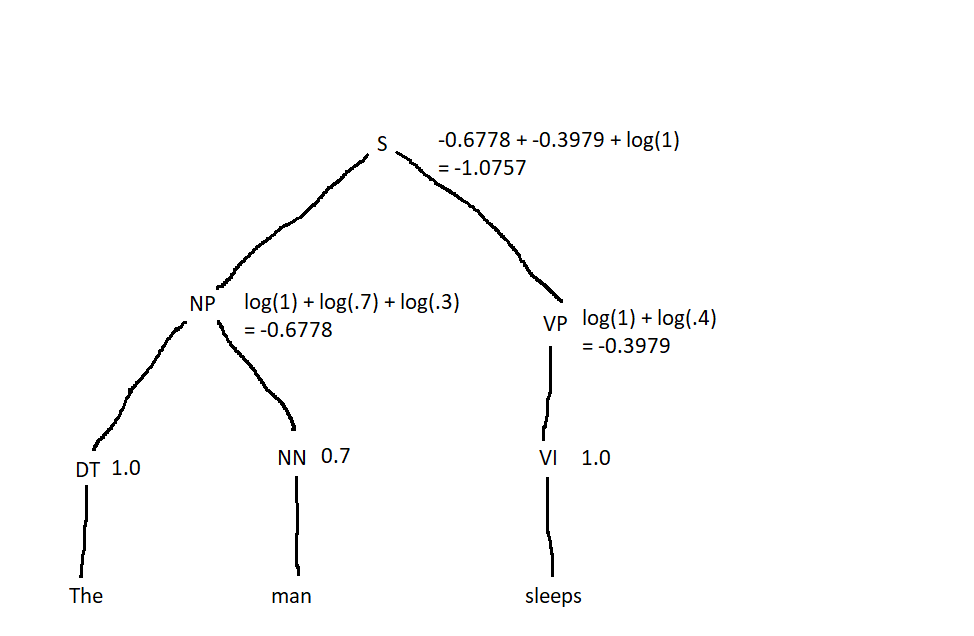
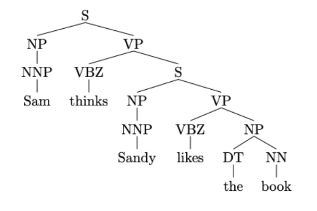
Natural Language Processing  
Homework and Programming Assignment 4

Parker Smith

11/2/2022

1. Given CFG grammar below.  
   Table

   Description automatically generated
   1. Parse the sentence using the above grammar and show parse tree.  
      Sentence: **The man saw the woman with the telescope**  
      **Map

      Description automatically generated with medium confidence**
   2. Show probabilistic parse tree with probability at each node in the tree for the following sentences.  
      Sentence 1: **The man saw the woman with the telescope**  
        
        
        
      Sentence 2: **The man sleeps  
        
      **
2. Show bracketed notation for the following tree:  
     
   [S [NP [NNP Sam]] [VP [VBZ thinks] [S [NP [NNP Sandy]] [VP [VBZ likes] [NP [DT the] [NN book]]]]]]
3. Consider the grammar G given by:  
   Text

   Description automatically generated  
   Validate your answer using CKY algorithm.
   1. Is w = **aaabb** in L(G)? No

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| 0 | A  [0, 1] |  |  |  | X  -> AT |
| 1 |  | A  [1, 2] |  | X  -> AT | S, T  -> XB |
| 2 |  |  | A  [2, 3] | S, T  -> AB |  |
| 3 |  |  |  | B  [3, 4] |  |
| 4 |  |  |  |  | B  [4, 5] |

The full word **aaabb** does not complete the algorithm with an S value, therefore **aaabb** is not within the grammar G.

* 1. Is w = **aaabbb** in L(G)? Yes

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | A  [0, 1] |  |  |  | X  -> AT | S, T  -> XB |
| 1 |  | A  [1, 2] |  | X  -> AT | S, T  -> XB |  |
| 2 |  |  | A  [2, 3] | S, T  -> AB |  |  |
| 3 |  |  |  | B  [3, 4] |  |  |
| 4 |  |  |  |  | B  [4, 5] |  |
| 5 |  |  |  |  |  | B  [5, 6] |

The full word **aaabbb** does complete the algorithm with an S value, therefore **aaabbb** is within the grammar G.

1. How are strings stored in spacy?   
   Strings are stored in spacy using a Doc object, which contains the all of the individual tokens within the string.  
     
   Describe the document similarity checking mechanism in spacy?   
   Spacy performs document similarity checking by using word vectors. Spacy first transforms all the words in a document to vectors. It then adds all of the vectors for each document together, resulting in a single vector for each document. Then, spacy uses cosine similarity to return a value from 0 to 1 to determine how similar two documents are to each other.  
     
   How would you compare similarity between two documents using spacy?  
   First, you have to use a model that includes word vectors (needed to check for similarity). Then, you create your two documents. Finally, you just use the .similarity() function to check the similarity of the first document with the similarity of the document used as a parameter within the function.  
     
   Please write down an example code for document similarity checking.   
   nlp = spacy.load(“en\_core\_web\_md”)  
     
   document1 = nlp(“I love nlp.”)  
   document2 = nlp(“I enjoy completing my nlp homework.”)  
   # This will print out the cosine similarity between document1 and document2  
   print(document1.similarity(document2))  
     
     
     
   What are the spacy pipeline component initiated while using default nlp().   
   The default spacy pipeline components are a vectorizer (tok2vec), a part of speech tagger (tagger), a dependency parser (parser), a rule based token attribute assigner (attribute\_ruler), a lemmatizer (lemmatizer), and a name entity recognizer (ner).  
     
   How would you add a custom component in the spacy nlp pipeline?   
   To add a custom component to the spacy pipeline, you have to first create a function that takes a doc object as a parameter and returns the modified doc object. Then you run the add\_pipe() function with the custom component function as a parameter. From there you can also determine where in the pipeline you would like the custom component to be.  
     
   Write a sample code which utilizes only two component tokenizer and add custom component in nlp pipeline.

nlp = English() # This returns an empty pipeline (Besides the tokenizer which is included by default).

@Language.component("num\_alphanumeric\_tokens")

def custom\_component(doc):

# I am inside my custom component.

alpha\_tokens = 0

for token in doc:

if token.is\_alpha:

alpha\_tokens += 1

print(alpha\_tokens)

return doc

nlp.add\_pipe("num\_alphanumeric\_tokens") # This adds the custom component.  
  
# Now, my pipeline consists of a tokenizer and a custom component in that order.

print(nlp.pipeline) # Visualize the pipeline

1. Go through the sequence of transitions needed for parsing the sentence “I parsed this sentence correctly”.  
   The dependency tree for this sentence is shown below.  
   
   1. At each step, give the configuration of the stack and buffer, as well as what transition was applied this step and what new dependency was added (If any).  
        
       **START STATE**Stack = [ROOT]  
      Buffer = [I, parsed, this, sentence, correctly]  
        
      **STEP 1**Stack = [ROOT, I]  
      Buffer = [parsed, this, sentence, correctly]  
      Transition Applied: Shift  
      New Dependency Added: N/A  
        
      **STEP 2**Stack = [ROOT, I, parsed]  
      Buffer = [this, sentence, correctly]  
      Transition Applied: Shift  
      New Dependency Added: N/A  
        
      **STEP 3**Stack = [ROOT, parsed]  
      Buffer = [this, sentence, correctly]  
      Transition Applied: Left Arc  
      New Dependency Added: parsed -> I  
        
      **STEP 4**Stack = [ROOT, parsed, this]  
      Buffer = [sentence, correctly]  
      Transition Applied: Shift  
      New Dependency Added: N/A  
        
      **STEP 5**Stack = [ROOT, parsed, this, sentence]  
      Buffer = [correctly]  
      Transition Applied: Shift  
      New Dependency Added: N/A  
        
      **STEP 6**Stack = [ROOT, parsed, sentence]  
      Buffer = [correctly]  
      Transition Applied: Left Arc  
      New Dependency Added: sentence -> this  
        
      **STEP 7**Stack = [ROOT, parsed]  
      Buffer = [correctly]  
      Transition Applied: Right Arc  
      New Dependency Added: parsed -> sentence  
        
      **STEP 8**Stack = [ROOT, parsed, correctly]  
      Buffer = []  
      Transition Applied: Shift  
      New Dependency Added: N/A  
        
      **STEP 9**Stack = [ROOT, parsed]  
      Buffer = []  
      Transition Applied: Right Arc  
      New Dependency Added: parsed -> correctly  
        
      **STEP 10**Stack = [ROOT]  
      Buffer = []  
      Transition Applied: Right Arc  
      New Dependency Added: ROOT -> parsed
   2. A sentence containing n words will be parsed in how many steps (in terms of n)? Briefly explain in 1-2 sentences why.  
        
      A sentence containing n words will be parsed in approximately 2n steps. This is because each word must be moved from the buffer to the stack, then each word must be assigned a dependency. It could take longer in certain circumstances if backtracking is needed.