Nickolas Whitman
Cs395
Assignment 2, Final
11/12/2020

Cloze Story Project

Introduction:

A question has been boggling the minds of computer scientists since the 70's and 80's, how do you get a computer to understand human language? Solutions to this question are through NLP (natural language processing) programs which are designed to make sense of raw text. There are many ways to implement this, but the best way is to have an NLP program learn off narrative chains to make sense of raw text. Narrative chains are groups of words that are linked to a protagonist/main entity. The more narrative chains the more learning opportunities the NLP program has, which can result in the program making more sense out of the sentence or story. Narrative chains are paired with a protagonist to help give a better idea of what is happening in the story. The whole purpose of having a narrative chain is to give your NLP program data to make sense of future text. "Narrative Chains: Link words in a list together into a sentence or a story. By using the words and associating them with each other you create a firmer connection between the new words and those already stored in your memory." This quote is another definition of what a narrative chain is, this definition is not from the Chambers and Jurafsky paper, here is the link to find the quote: http://www.nflrc.hawaii.edu/prodev/SI99/vocabulary/narrativechain.htm#:~:text=narrative%20chain,already%20stored%20in%20your%20memory.

Related Work:

To test the comprehension of your NLP program you should use a cloze test. A cloze test is where you delete every nth word from raw text to test if your NLP program can fill in the missing words to make sense of the text. A paper from 2018 called "Chengyu Cloze Test " by Zhiying Jiang, Boliang Zhang, Lifu Huang and Heng Ji from RPI (Rensselaer Polytechnic Institute) describe the process of creating their own cloze test. Here is a row from the graph from section 4.1 of their paper (link to the paper is at the bottom of this paragraph):

TYPE	Query	System	Ground Truth	Analysis
Discourse Coherence	人们面临灾难，不得不____，离开他们自己的村落。 When facing disasters, people had to _____ and leave	逍遥法外 at large	背井离乡 leave one's hometown	Our system focused on the shared meaning of escape/leave while ignoring that Chengyu has a specific object "the arm of the law".

	their own villages.			

This paper dove into the components of the researchers' Cloze Test algorithm. The data table above shows that their algorithm resulted in placing "逍遥法外 " (which translates to "at large") into the missing text segment indicated with "____". The actual text that was removed was "背井离乡" which translates to "leave one's hometown". This doesn't make sense if you insert the english words into the actual query but I assume that the english translation is slightly wrong which can confuse the reader. Anyways, the whole point of this data table is to show how an algorithm can return words to try and make sense of the story or sentence. The goal of this project is to implement something like what the RPI students who wrote the "Chengyu Cloze Test" paper. The Cloze Test version I created is an algorithm that looks at the given text and returns a probability of an ending that makes sense of the story. This paper worked with Chengyu which are traditional Chinese expressions and I plan to work on a similar path they did except with English. This cloze test paper is the paper I would like to highlight as related work to what I am doing. (https://blender.cs.illinois.edu/paper/chengyu2018.pdf)

Method:

My goal is to create a program that can look at two ending sentences and choose which one is the correct ending sentence. We have a data set that is a story of 4 normal sentences and 2 ending sentences. Professor Bose was gracious enough to give us some code to use in our program. Here is the code for my Cloze Story:

```
import chains
from pprint import pprint



def parse_test_instance(story):
   """Returns TWO ParsedStory instances representing option 1 and 2"""
   # this is very compressed
   id = story.InputStoryid
   story = list(story)
   sentences = [chains.nlp(sentence) for sentence in story[2:6]]
   alternatives = [story[6], story[7]]
   return [chains.ParsedStory(id, id, chains.nlp(" ".join(story[2:6]+[a])),
*(sentences+[chains.nlp(a)])) for a in alternatives]

def story_answer(story):
   """Tells you the correct answer. Return (storyid, index). 1 for the first ending, 2
for the second ending"""
   #obviously you can't use this information until you've chosen your answer!
   return story.InputStoryid, story.AnswerRightEnding

# Load training data
```

```
data, table = chains.process_corpus("train.csv", 1000) # train words on train.csv and
build table to use in determining answer.
#print(table.pmi("move", "nsubj", "move", "nsubj"))

# load testing data
test = chains.load_data("val.csv")
n = 0 #values for testing performance. n is num right, tn is total count
tn = 0
for t in test:
    one, two = parse_test_instance(t)
    one_deps = chains.extract_dependency_pairs(one)
    two_deps = chains.extract_dependency_pairs(two)
    #pprint(one_deps)
    #pprint(one[2:])
    #pprint(two_deps)
    #pprint(two[2:])
    #Look at dependency list and find out which sentence offers more verbs to the main
entity (protagonist). More relevance means more coherent answer.
    #0 is the first entity, 1 is the second entity, etc.
    if (len(one_deps[0])>len(two_deps[0])):
        print("my answer: 1")
        if(t.AnswerRightEnding == 1): #check if the answer is correct
            n+=1
        print("right so far :"+str(n))
    elif (len(one_deps[0])==len(two_deps[0])):  #if dependency lists are the same size
for the main entity, look at the occurances of the words from the test data and choose
one that occurs most
        total = 0
        for (w1,w2) in zip(one_deps[1][0],two_deps[1][0]):  #iterate through first
entities
            word1 = w1[0]
            verb1 = w1[1]
            word2 = w2[0]
            verb2 = w2[1]
            total += table.pmi(word1, verb1,word2, verb2)  #return the most occuring
instances of the two versions of verb dependencies.
        if(total <= 0):
            total = 1
        else: #if total is 0 then the words are the same.
            total = 2
        if(t.AnswerRightEnding == total):
            n+=1
        print("right so far :"+str(n))
    else:
        print("my answer: 2") #if the second dependency list is larger than it contains
an extra centence that fits in the story, thus the 5th sentence is correct.
        if(t.AnswerRightEnding == 2):
            n+=1
        print("right so far :"+str(n))
    # logic to choose between one and two
    tn+=1 #increment after each round to keep track of total stories tested.
    print("total"+str(tn))
    pprint("answer:"+ str(story_answer(t)))

print(str((n/tn)*100) + "%") #percentage of correct decisions. For testing purposes
```

This program was able to get 51.496% of the ending stories correct. I consider this a good score because in the Winter 2018 results for the story cloze test I would have placed 5th in the competition. The submission by the username ‘ROCNLP’ scored 50.4137% for 5th place in the winter 2018 Cloze test competition. To get a decent score like this there were several main focus points I had to look at. The first point I would like to make is that the dependency list is a list of verbs (narrative chains) that are dependent on the main entity (protagonist). I looked at both dependency lists for each story with separate ending sentences and returned the one that had the biggest dependency list. This is because the last sentence would have to be not as relevant to the story if it isn’t talking about the protagonist, and you can tell that the last sentence is relevant because it will add a verb to the dependency list of the main character (entity 0). Another point I had to explore is the fact that some stories can have both ending sentences talk about the protagonist, resulting in the size of the protagonist dependency list to be the same on both options. For this case I ended up looking at each word in the dependency list and calling a pmi function call on each verb/word pair on both lists. If the total pmi was negative then the first list would be more relevant because statistically the words appear more in the training set. If the total pmi was positive then the second option would be chosen for the same reason. Testing this was simple, I just kept track of stories that were correct and the total number of stories looked at. The number correctly identified divided by the total number of stories returns the percentage of occurrences that my algorithm predicted the correct story ending ((n/tn)*100 = %correct).

Evaluation:

What I found interesting from my 51.496% success rate is that guessing could end up getting you 50% correct. I see that my algorithm made a little impact over just guessing which sentence fits better, so I consider that a success. Also my results would have made it to the top 5 leaderboard of the Winter 2018 Cloze Story Test on CodaLab. This is cool because my algorithm is on par with some of the code other people wrote. I evaluated my code on only 1000 stories from the training set. The training set is called train.csv and the story set that I tested my code on is called val.csv. I am sure if I evaluated on some more 15,000 stories my algorithm would have performed better. The only reason why I did not do this is because the training function takes a long time to finish computing. The 1,000 stories took about 2minutes to compute and I did not want to wait 30minutes to compute all the stories in the train.csv file.

Summary:

In summary I created a Cloze Test algorithm that is able to correctly predict more than 51% of the random final sentences from a story. It focuses on looking at the dependency lists for each possible ending sentence with the previous four sentences and looks to see which list has more dependencies. If the dependencies are the same then my algorithm will look at the frequencies of occurrences of the dependencies in the training set and choose which dependency is more likely to occur. This proved to be an average algorithm that barely beats an algorithm that guesses randomly (50%). Here is a list of all the packages and libraries I downloaded: cython, pandas, neuralcoref, spacy, and tqdm. After installing those packages download the files uploaded on this GitHub: https://github.com/mrmechko/narrative_chains

Extra Credit1:

My initial work including just simply returning the option which contained the larger dependency list for its story. I expanded upon this and included an option which contained the possibility that both sentence options were included in entity0's dependency list. This proved to be more difficult than I hoped for, yet it help improve my system's performance.

Extra Credit2:

Link to gitHub repo: https://github.com/NickolasDesigns/CS395ClozeStory