# COMP5211 Report

ZHOUYingqi,20256851

The source code contains five directories. I convert the collection of Winograd Schemas to txt and use Stanford-corenlp to do the analysis. The output of Stanford-corenlp is used as input for feature1, feature2 and feature4.

Feature1 is narrative-chain feature. Basically, it solves the following problem:

Ed punished Tim because he tried to escape.

The first step is to identify the event which two candidates involved and their roles in this event. (punish-s,Ed) and (punish-o,Tim) are the two tuples generated from this sentence.-s denoted the subject.-o denotes the object. Then we identify the event the pronoun participates and its role, in this example (escape-s,he) and (try-s,he) are generated. Finally, we search the narrative chains (available from http://cs.stanford.edu/people/nc/schemas/schemas-size12.) to find the line which contains both the target event and candidate event. In this example, we search the line containing escape-s and punish or try-s and punish. And the search result is [ give-o deny-o decertify-o escape-s decertified-o cooperate-s rule-o declare-o certify-o visit-o allow-o punish-o ] So we know we should find the person with punish-o, that is Tim.

Feature4 is polarity determined feature. It is used to solve the following problem:

The town councilors refused to give the demonstrators a permit because they feared violence.

The town councilors refused to give the demonstrators a permit because they advocated violence.

The man stole the neighbor's bike because he needed one.

The police captured the criminals because they were less prepared.

The police captured the criminals because they were not prepared.

The police captured the criminals because they were more prepared.

Firstly, we identify the target event and candidate event as previous feature. In the first example is refuse and fear. Then we search Wilson et al.'s (2005b) subjectivity lexicon

to find whether these words have positive or negative meaning. In this example, refuse is negative and fear is negative as well. The polarity of these event corresponds to the polarity of their subjects. So, “they” are resolved to councillors. Then we need to consider negation(not), discourse connective(although,but) and less because they negate the polarity of the target event.

Feature2 is google search. We extract target event and candidates, pair them as query to perform search. For example, to solve the following question:

The knife sliced through the flesh because it was sharp.

We pair knife and flesh with sharp separately, so we generate “knife sharp” and “flesh sharp” as two queries and count the number of the results returned. Then we compare the count and resolve them to the candidate with higher count.

Since google search is kind of random so we firstly adopt feature1 or feature4. When we occur the problems unsolved, we seek for google search. Finally, the result is total 324 questions(some lines are separated by Stanford-nlp so the number is increased, but I ignore the sentence with less than 8 tokens.) The program correctly solves 99, leaving 121 unsolved.

How to run the source code:

I should apologize because I didn’t write a user-friendly interface. In the integrate directory, challenge.txt is the file copy from the 144 Winograd schemas xml input. Then challenge_processed.txt is the file containing the processed challenge to serve as the input to Stanford-nlp. And input1.txt,input2.txt,input3.txt correspond to the /feature1/out.txt,/feature4/out.txt and python_file/output.txt. All the input.txt in feature1,feature2 and feature4 directories correspond to the output from the Stanford-nlp.

Instructions:

Step 1: copy text from .xml page and paste it to /integrate/challenge.txt

Step2: Run integration.cpp to process the challenge.txt file , processed result is stored in challenge_processed.txt.

Step3: copy challenge_processed.txt to standford nlp directory and rename as input.txt,run this command line.

```
java -cp "*" -Xmx2g edu.stanford.nlp.pipeline.StanfordCoreNLP -annotators
tokenize,ssplit,pos,lemma,ner,parse,dcoref -file input.txt -outputFormat text
```

Step4: copy the content of input.txt.out to the /feature1/input.txt, /feature2/input.txt and/feature4/input.txt, run their cpp files.

Step5: copy the /feature2/queries.txt to /python_file , and rename it as queries.txt

Step6: run google_search.py

$python google_search.py

Step7: copy /python_file/output.txt to /integrate/input3.txt, copy/feature1/out.txt and /feature4/out.txt as input1.txt and input2.txt in the integrate directory, and then run the integration.cpp. Result is printed on the console.