Interim Progress Report

Haya Al-Kuwari, Akhyar Kamili, Mohammed Nurul Hoque, Abubaker Omer

November 21, 2018

This report presents our group's progress in the project and our implementation plan so far. We experimented with Stanford CoreNLP and decided to use its tokenizer, tagger and dependency parser. We also decided to implement parts of stanford's `semgrex` ourselves. Finally we modified and simplified our pipeline as described below.

1 Initial Experiments

We experimented with Stanford CoreNLP library using simple sentences. We found the idea of using `tokregex & semgrex` to generate and answer questions particularly appealing because it simplifies a lot of processing as regex matching-like operations. Unfortunately, the CoreNLP server takes up huge amount of space and time when these features are used, while it is quite fast when used just for regular operations like tokenizing, tagging and parsing.

2 Tested Prototypes

We implemented two modules to deal with question generation. Firstly, we implemented the 'Document' class, to simplify working with text documents in which we only need to supply its method `generateQuestionsFromPattrens` with a list of tuples of pattren and function for createing questions from these matchings. We, also, implemented a class `DepGraph` which is a subset of the functionality of stanford's `semgrex`. It takes relations of the form "node x has tag *VBD* and is a governor of a *nsubj* relation to a node y that has tag *NNP* and also a *dobj* relation to a node z of unspecified tag" (In CoreNLP notation `{tag:VBD} >nsubj {tag: NNP} >dobj {}`). It returns indices of the words matching the nodes in the same order. The current implementation matches only in a single level but we will expand it to match recursively on the parse tree.

3 Results

We tested out implementation with an actual template and document. The listing below shows some of the results.

4 The Pipeline

4.1 ask

We split the document to sentences using class `Document` which uses CoreNLP's `ssplit`. Then randomly sample sentences with bias to increase probability of important sentences (e.g. 1st sentences in paragraphs).Then we search for our templates in the sentence and output the resulting question if found.

4.2 answer

Given a question we first filter the sentences that have very low similarity. Then for the remaining sentences we search for a node that has the same dependency path as the question word (When, What, ..)

and output the text under that node. If not found, we fall to a fall-back method that just outputs the sentence with maximum matching.

../src/test.py

```python
#!/usr/bin/python3

from depgraph import DepGraph
from utils import Document
from pycorenlp import StanfordCoreNLP

client = StanfordCoreNLP('http://localhost:9000')

with open('../data/set1/a2.txt') as f:
    doc = Document(f.read())
print(len(doc.sentences))
for sent in doc.getSentences():
    dg = DepGraph(client, sent)
    res = dg.match('VBD', [('nsubj', None), ('dobj', None)])
    if res:
        verb, subj, obj = res
        print('Who {} {}? {}'.format(dg.tokens[verb], dg.tokens[obj], dg.tokens[subj]))
```

```
Who won Cups? Donovan
Who reached quarter-finals? team
Who raised him? mother
Who allowed him? mother
Who attended School? He
Who attended Academy? Donovan
Who signed contract? Earthquakes
Who had trouble? Donovan
Who had success? Donovan
Who scored goals?...
```