Runqian Huang (rh627)
Bjorn Bjornsson (bgb64)
James Kaye (jlk298)
Sherbin Abraham (sa946)

Project 3 Part 1 Report

Baseline Implementation

Description of Final Implementation:

We implemented a sliding window baseline, similar the one mentioned in the Critique 3 paper. We calculate the unigram overlap between the question and the possible sentences that may contain the answer. We begin by iterating over all paragraphs and separating the context into sentences (to reduce the size of the answers). Then, we keep track of which sentence had the most overlap (i.e. similar tokens) with the question. If we find a sentence that has a greater overlap with the current questions, we find the start and end of the overlap. We then use the part before and after the overlap portion in order to reduce the overly verbose answers.

Justification of Implementation:

We decided on a sliding window implementation mainly because we needed to account for the fact that the answers could be of variable length. In addition, we also attempted another implementation using POS tags and word vectorization; however, the variable length answers made this implementation difficult to realize. It seemed that in the other attempts, we chose a fixed length for the answer. This wasn't dynamic enough, which is why we decided on the sliding window idea. Furthermore, bigram holds a certain order to the sequence of tokens. However, we wanted the tokens that showed up in the question to be dynamic. For example, it could be the case that the order of some lexeme in the question is reversed in the correct answer. Therefore, a unigram overlap was the better choice.

Evaluation of Result:

When running the baseline on the **development** set:

- ➔ *Exact Match* = 5.695
- ➔ *F1 Score* = 21.950

When running the baseline on the **training** set:

- ➔ *Exact Match* = 4.351
- ➔ *F1 Score* = 19.444

When running the baseline on the **testing** set:

- ➔ *Exact Match* = 4.037
- ➔ *F1 Score* = 17.984

The output had a lot of blank answers, which really hinders our F1 score. This may be occurring because of the way the sentences are being parsed. We split the sentences by determining if a punctuation mark is found. This may cause much smaller sentences to be chosen for the most matches and the entire sentence being stripped (since we choose the tokens before and after the overlap section). In addition, this implementation does not account for multiple sentence reasoning nor syntactic/lexical variation in the answers and questions.

Runqian Huang (rh627)
Bjorn Bjornsson (bgb64)
James Kaye (jlk298)
Sherbin Abraham (sa946)

Proposal for Final System

The first part of our implementation will be to transform terms into word vectors so that we can deal with synonyms (using cosine similarity). We have implemented this in previous projects, so we will use word2vec or GloVe to turn words into word vectors. We will then proceed to tag tokens with their respective POS and NER tags using python's built in nltk package. This will come into play when we classify both questions and answers.

We proceed to manipulate the question formulation and classification. We will classify the questions based on their *wh-word*. Based on this cue, we know the type of answer we need (eg. "where" is usually associated with a location, "who" with a person, etc.). As outlined in the textbook, we can develop a mapping from questions that do not have a *wh-word* to questions that do. This is where the tagging will help, because we can find the cosine similarity of a sentence with the question and then proceed to look for the POS that corresponds with the question.

We will also try to add extensions such as pattern matching and keyword distance to improve our model. We can try these methods individually or in some combination. As a stretch goal, we could also implement the logistic regression model laid out in the paper along with our model to improve it further, however this may be an offshoot to our actual model. We will try to optimize on top of the baseline. If we find ourselves reaching a ceiling with the sliding window method, we will proceed to another implementation.

Member Contributions

- Runqian Huang - Design and wrote the baseline code *baseline.py* that uses sliding window method focusing on the unigram overlap.
- Sherbin Abraham - Wrote alternative version of baseline (didn't work) that focused on word vectorization and POS tagging (will be used in final version).
- James Kaye - Tested and analyzed output of *baseline.py*.
- Bjorn Bjornsson - Wrote the write up.