11-791 HW 2 Report

Chenyan Xiong
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
cx@cs.cmu.edu

1 General Design Idea

In general, based on the given type system, the answer scoring pipeline should include the following steps:

1. Question and answer reader: read the question and answers in the given input text.
2. Tokenizer: tokenize the input text.
3. Ngram Annotator: annotate the tokenized text into ngrams.
4. Question-answer pair scorer: given a score to a question-answer pair based on some strategy.
5. Result evaluation: evaluate the performance of scored answers.

In these steps, step 1: Question and answer reader could either be run in the beginning, or after tokenizer and Ngram annotator. The only place where we have more flexility in design is the question and answer pair scorer, which we will discuss several possible simple models in next section.

2 Possible Models

QA system is a big research area, but for this homework, there are several very basic ones to consider:

Vector Space Model. We could express the question and answer with vector space model. The dimensions could be tokens or ngrams. We could then using the cosine similarity between question's and answer's vector space models to assign a score to a question-answer pair.

Language Model. Language Model is a widely used method in IR fields. It could also be used here to rank the questions based on their language models' probabilities in generating questions. The language model could also be trained on tokens or ngrams. However, in this homework, we have so few training corpus, which makes the essential part: smoothing in language model very hard to work. One solution is the add some external corpus to this project and calculate the corpus probability of a ngram on the external corpus. However, as the answers' term distribution may vary with external corpus, it is not sure whether it will help ranking.

Supervised Ranking. One could also use supervised ranking to train a model (for example, RankSVM) on given training data, and learn the weight to combine different features. Features could be ngrams, or hyper features, like the score of language models. However, as there is only 2 questions available, it is impossible to learn a suitable model in such small amount of training data.

Our Choice. As there is too few training data and corpus information, it seems to us the most simple method vector space model is the best choice. Thus we implemented cosine similarity on vector space model. Further more, we use three ways to construct vector space: unigram, bigram and trigram. The cosine scores are calculated from three separately and the final score is linearly merged with weights manually assigned as :(unigram, bigram, trigram) = $0.4, 0.4, 0.2$.

3 Detailed Implementation

In this section, we discuss the detailed implementation of our classes:

1. Tokenization: this class takes the JCas as input, and call stanford nlp lab's tokenizer to tokenize raw text in JCas to tokens, recorded as annotations.
2. QuestionAnswerReader: this class segments the input raw text in aJCas into question and answers, based on format given by homework instructions. It will add annotation to JCas as question and answers.
3. NGramAnnotator: this class annotate ngrams to the questions and answers of JCas. The n could be set by parameter 'ngramn'. There could be multiple n's in the parameter, and this class will annotate all ngrams.
4. CosineScorer: this class assign scores to annotated question-answer pairs. The score is calculated as discussed in previous session.
5. ResultAnalysis: this class evaluates the precision@N of CosineScorer's output. The collection average precision are also recorded and output in the end.

For more details in implementations, please refer to the java docs.