CS5740_P3_Question Answering_Report

Pengcheng Zhang
pz84@

Jiangjie Man
jm2559@

Puran Zhang
pz75@

1 Introduction

In this report, we describe how our Question-Answering system was implemented and compared its performance with the baseline system we built in part I. For our QA system in part II, we use NER tagger for answer type detection and build a logistic regression model for extracting candidate sentences and extracte several types of features for each candidate answer. Features we choose to incorporate are the "TF-IDF score" and "keyword distance" as well as the "number of matching keywords" to help our final model pick the correct sentences. We also include the evaluation and analysis of our QA system's performance compared with our baseline system.

2 Baseline System

Our baseline system is consist of several modules, a Question module, an Answer module and a last module in charge of producing and outputting the answers. The Question class defines a question object which can be used to parse, tagged and classified. Every question is a string obtained from input files. The Answer class gives 5 guesses and 5 document ids ranked from top to the fifth for each answer. It uses nil as the answer-text if the system finds no answer for a particular question. The answer-text should be 10 or fewer words.

In particular, when pre-processing the data, after valid text is extracted from the raw document file, we extracted "keyword" – noun phrase (NP) from the question using NLTK's pos-tagger. To determine the frontiers of this noun phrase, we define a local grammar for the NP in English, such as "Adjective + Proper Noun".

Then, for each question, we matched the keywords with text data in the documents by sentences. If the candidate sentence contains keywords, we striped all the keywords in the sentence and then extracted NPs in the sentence. All these NPs are stored in a frequency dictionary where we sorted the top 5 NPs to be our guesses for answer.

3 Final QA System

Based on the performance of our baseline system, which is a naive matching-word system, we decided to explore machine learning methods/models. In the baseline, we deployed hand-written rules to extract answers. However, these rules are very time consuming. We cannot merely learn what we need to perform classification from examples since rules are hard-coded. Besides, when new question types arise, the system is not able to handle them appropriately. New rules must be written by hand to cover these new types. In addition, if it is decided that we need to use a new set of answer types, it is probable that many rules will need to be rewritten. Finally, it is even more difficult to extend the answer types to more specific types.

We incorporated the tf-idf, which is a weighting factor that reflects how important a word is to a document in a collection or corpus, to help our final model pick the correct sentences. In addition, we used length features to bias our model towards picking common lengths and positions for answer spans. Span word frequencies were integrated to help us bias the model against uninformative words. Furthermore, according to the paper on SQuAD (Rajpurkar, et al., 2016), we used constituent label and span POS tag features to guide the model towards the correct answer types.

3.1 Overall Architecture

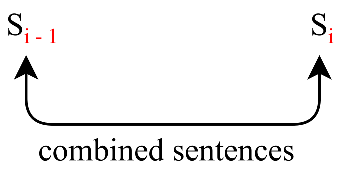
In this system, we built our logistic regression model for predicting answers with the training sets (in the "doc_dev" folder) in baseline phase. The training process is listed below:

- **Processing question texts**
  - First, we did the answer type detection by simply extracting and classifying question words (who, when, where) in those questions in our training sets;
  - We then extracted keywords from question sets, preparing for subsequent matching tasks. In the keywords extraction algorithm, we followed selecting all non-stop words and then all NNP words in recognized named entities;
- **Extracting candidate answers**
  - We used Stanford Named Entity Recognizer[1] to label sequences of words that are named entities in candidate texts;
  - Then we extracted features (name them X) such as tf-idf from these candidate answer texts;
- **Matching pattern.txt with candidate answers**
  - With extracted candidate answers, we compared/matched them with pattern.txt, which is the actual answer. Specifically, we matched each of the candidate answer with the regular expression in pattern.txt and scored (candidate answer) with 0 (as dis-matched) and 1 (as matched). We name this as Y;
- **Training with Logistic Regression Model**
  - Now with X and Y, we trained our logistic regression model that predicts answers with inputing features (X) and true label (Y) using `sklearn` package. Specifically, We first convert both X and Y into `DataFrame` and then use sklearn.linear_model.LogisticRegression `fit` function to train our model.
- **Testing**
  - After training, we can predict which candidate answer is the correct answer according to our logistic regression model. The testing process is similar to training, but we only extract features from candidate answers which are selected by NER tagger in the test documents.

[1]http://nlp.stanford.edu/software/CRF-NER.shtml

3.2 Features

- **TF-IDF**:
  TF-IDF, short for term frequencyinverse document frequency, is considered as a feature of our system. Since this value has been given in each document, we haven't calculated it our self. Instead, we just used the score in each document.
- **Question keywords**:
  It's the number of question keywords in a single candidate sentence. We think it may be a good indicator to select candidate answers.
- **Keyword distance**
  It's the distance in words between the candidate and query keywords. Since candidate answer is extracted from document, it's guaranteed that candidate will be located in current sentence. However, keyword may or may not exist in the document. Here we only consider the distance of a keyword if it's in current sentence or neighbour sentence as shown below.



We define it as a “combined sentence”. If a keyword doesn’t exist in the combined sentence, we will assign a considerably large distance to this keyword and candidate answers. After calculating all distances, its reciprocal will be summed as a feature. Although (Lin, Chuan-Jie and Chen, Hsin-Hsi, 2001) claimed that for duplicate candidate answers, only the one that’s nearest to key words should be used to calculate distance, we think all candidate answers are important attributes of our system so even if there exists duplicate answers, distances from all of them will contribute to our system.

3.3 Preprocessing texts

In preprocessing stage, we found the format of data is TRECTEXT, which is similar to XML and HTML. For the simplicity of our work, we use BeatifulSoup4[2] to parse the document file.

After investigating the document file, we found content within the following tag is useful to out baseline system (Ian, 2002).

- `<TEXT >...</TEXT>`
  The body of the article. The exact structure of the body text varies from format to format.

- `<LEADPARA>...</LEADPARA>`
  The lead paragraph of the article.

- `<LP>...</LP>`
  Similar to LEADPARA

- `<P>...</P>`
  We found there are some special documents that do not contain any <TEXT><LEADPARA><LP>. However, text within <P>...</P>still contain some valid information within the <P>...</P>, if our parser cannot extract any valid information from tags above, content from <P>...</P>will be extracted.

[2]https://pypi.python.org/pypi/beautifulsoup4

As mentioned in 3.1, firstly we use Stanford NER tagger to extract all candidate answers. However since the tagger is relatively slow, it’s not feasible to run the tagger every time before training our model. Instead, we serialized NER tagged sentences in NER_doc_dev and NER_doc_test folders so that the program can load NER tagged data directly. Folder structure is the same as doc_dev and doc_test, except that now serialized objects are stored in each document files. We’ve provided instructions for how to access these data files in readme.md.

What is more, multi-process is used to spped up our program. Firstly all questions are extratced from questions.txt, then they are divided into k parts (where k is the number of cpu core according to the computer spec), then k subprocess will be used to extract answer separately. In order to avoid writing confilts, an extra subprocess will be used solely for the writing task.

3.4 Implementation Details

For example, we have a question which is “Who is the leader of India?”. First, we detect what type of question it is. Using simple parsing method, we find the stopword ’who’ for this question, thus its type is ‘PERSON’. So our answer type would also be ‘PERSON’. Since we also want to extract the keywords contained in this question through our handwritten rules, our keywords are ‘leader’ and ‘India’.

Then, we start NER-tagging with `Stanford NER` package and try to find all the candidate answers – unigram and bigram words whose tag is ‘PERSON’ through a `for` loop. Candidate answers may include ‘Rao’, ‘Rao Hans’, ‘Gandhi’, ‘Gandhi Jawalharlal’, etc. In the sentence where candidate answers reside, we extract features using `extract_distance` and `extract_count_query` functions. These features are stored in lists and we have to transform them into Dataframes as mentioned in 3.1. Then, with our machine learning model, we use `predict` function and get the list of 0s and 1s, which tells us which candidate words can be a real answer if it is a '1'. Finally, we output our 5 guesses for each question into `answer.txt` file.

4 Result Analysis

4.1 Baseline Performance

	Mean reciprocal rank
1	.129
2	.122
3	.128
4	.128
5	.122
avg	.126

The table above shows the performance of our baseline system after running 5 times, using *mean reciprocal rank* as the main metric for evaluation which is a statistic measure for evaluating any process that produces a list of possible responses to a sample of queries, ordered by probability of correctness.

The result is relatively low compared to human performance. The reason for this would be over-identifying NP. For example the question is "When was Hurricane Hugo?", when finding answers, two words in the same category in a candidate sentence can be both recognized as NP, e.g. "Hurricane Katrina" and "Hurricane Sandy", but neither of them are the correct answer of this question.

4.2 Final QA system Performance

	Mean reciprocal rank
1	.074
2	.069
3	.071
4	.074
5	.074
avg	.073

Scores of the final system is shown above in the table. With incorporating features including *tf-idf*, *question keywords* and *keyword distance* into our regression model, we didn't get improvement in terms of the mean reciprocal rank comparing to our baseline system (even poorer than the baseline). This result is quite stunning and thus intriguing for our team.

First, the machine learning approach requires a relatively large training sets to build the model. For this project, in our training process, we have about 20000 files, which we thought would be enough for building our model. Relationship between features and binding correct answers were not well-reflected in our model. Thus, the output from the model were not ideal.

In addition, we thought it might due to that our features extracted from candidate answer texts are not well-chosen, well-combined and not enough. These features may just be basic features for general QA system and may not perform well on this specific question set. We've experimented a few combinations of these features (T: tf-idf; K: question keyword; D: keyword distance):

features	Mean reciprocal rank
T, K, D	.074
T, D	.069
T, K	.069
T	.064

And obviously none of them seems to be strongly effective for this task.

We also think that the "matching type" in our candidate answer extracting process is too general. For example, if a "When" question is being raised, and it is asking for a "year". For our system, when doing NER matching, we only specify tags to "DATE" instead of "year" and "month" and etc. Therefore, any time-related type text, which is too general for this question, will be selected as our candidate answer.

What's more, our system give answers to questions that couldn't find an answer. For example:

```
<top>
<num> Number: 212
<desc> Description:
When did the American Civil
War end?
</top>
```

The answer is "1865". While "1865" doesn't exist in any document under the folder

/doc/dev/212. It might be due to that the document set only contains the top 100, answers might exist in document that's ranked outside the top 100. Since our answer has been ranked, one solution to avoid this is to set a threshold score in answer list and drop answer that is below the threshold value and return a nil even if there exist valid answers in our answer list. We've tried this idea, however MRR of the system is even worse because now the system cannot give answer to questions that it was able to answer. It might be due to that heuristic score ranking function of our system is not good enough to handle this situation(in other words it can be used to rank answer but the score might distribute unevenly so it's difficult to set a threshold value). However, considering that designing a score function that will give an even distribution of score may affect the performance of ranking, and only a few questions don't have a answer(i.e. 116, 175, 212...) in document but exist in pattern.txt. We ignore this case.

It should also be noted that, questions such as 'who..' may expect a different kind of answer. For example

```
<top>
<num> Number: 93
<desc> Description:
Who is the leader of India?
</top>
<top>
<num> Number: 94
<desc> Description:
Who was Whitcomb Judson?
</top>
```

Number 93 expect a a name of the leader of India, while question 94 actually expect a definition of Whitcomb Judson. First of all, a trivial consideration is our system should never give a answer that occurs in the question. So we should avoid

```
Q:Who was Whitcomb Judson?
A:Whitcomb Judson.
Correct:Zipper.
```

However, our system still fail to answer questions such as # 94 even if we've filtered what is mentioned above.

After investigating the answer, we find the correct answer is actually very close to the question key word.

```
...and the zipper
(by Whitcombe L. Judson).
So much...
```

Thus we tried to design a different distance function such as a modification of sigmoid function. However the performance doesn't improve so we've dropped it.

5 Team Member Contribution

Our work of distribution is listed as follow:

- Pengcheng Zhang: Text parsing and feature extraction
- Puran Zhang: Feature Selection and Error Analysis
- Jiangjie Man: Logistic Regression model and Answer extracting

References

Rajpurkar, Pranav and Zhang, Jian and Lopyrev, Konstantin and Liang, Percy. 2016. *SQuAD: 100,000+ Questions for Machine Comprehension of Text.* arXiv preprint arXiv:1606.05250 (2016).

Roberts, Ian 2002. *Information Retrieval for question answering* (2002).

Lin, Chuan-Jie and Chen, Hsin-Hsi 2001. *Description of NTU System at TREC-10 QA Track.* (2001).