CMSC723 When to Buzz Project Report

Zheng Xu and Weiwei Yang

December 18, 2014

1 Problem Description

We work on the Kaggle project, when to buzz, which is a question answering problem generated from quiz bowl. For each document in quiz bowl, we have the question ID, question text, sentence position, category, guesses and scores generated by QANTA and IR Wiki. We also know the correct answer for training.

Our goal is to select a guess for each question as the final answer. We are permitted to use any external code and data except quiz bowl questions.

2 Approach

2.1 Preprocessing

In preprocessing phase, we first crawl Wiki text of all guesses provided by QANRA and IR Wiki. We then apply stemming on the crawled Wiki text and the provided question text. We apply POS tagging and try to filter words by POS, but it seems doesn't help to improve the performance.

Apart from techniques above, we take away general common words (a, the, and, etc), as well as specific common words in Wikipedia (like references and websites).

2.2 Get More Information

Our idea of selecting answers is to calculate the score for each guess and choose the one with highest score. QANTA and IR Wiki scores are very useful, but they are not enough. Therefore, we compute two more kinds of scores on our own.

2.2.1 Bag-of-Words Cosine Similarity

Bag-of-words (BoW) is a basic word-level technique to convert each document into a vector. Each dimension of vector denotes a word and the value of each dimension is the frequency of word in current document.

The intuition is that if a guess is likely to be the answer, its Wiki text is more likely to use similar words with question text. Thus, we compute the cosine similarity between BoW of question text and Wiki text of each guess.
2.2.2 Topic Vector Cosine Similarity

Besides word-level scores, we try to compute document-level scores. One of the best way to represent a document is to analyze its topic distribution and get topic vector. Naturally we adopt topic model and implement latent Dirichlet allocation (LDA) [3]. We run LDA on all question text and Wiki text and get matrix $\theta$ which contains topic vector for each document. With topic vectors, we compute cosine similarities between each question text and each of its guess Wiki text.

2.3 Linear Combination

Since we want to compute a final score for every guess, our idea is to linearly combine the four scores (see Equation 1) of each guess and select the guess with highest score as final answer.

$$score = a_1 \times QANTA + a_2 \times IRWiki + a_3 \times BoW + a_4 \times TV$$

We set $\sum a_i = 1000$ and try all integer combinations of four parameters. The combination which performs best on training set is selected and apply on test set.

2.4 Training by Categories

As all questions belong to only four categories, we come up with the idea that we can train a model for each category. The intuition is that it is difficult for a general model to fit questions in different categories. It may get some questions wrong in a category in order to get more right in another one. So we split the training and test set by categories and train a model for each of them.

2.5 Classification Approach

In this part, we give an alternative approach. We utilize a binary classification framework to handle the problem. For the candidate guesses of each question, the labels are either right or wrong. The feature for each guess includes question part (each category is an indicator, sentence position) and score part (for each of the four scores in Sec. 2.3, we use the score, the normalized score, the rank of the guess with respect to the score for each guess). We train a SVM classifier for all the data, as well as separated classifiers for different categories and different sentence positions. We linearly combine the output scores of SVM classifiers.

3 External Code and Data

3.1 Wiki Pages

In order to better understand guesses, we crawl all guesses’ Wiki page using Wiki API [2].

3.2 OpenNLP Package

We employ OpenNLP package [1] to stem and tag POS of question text and Wiki content of guesses.

3.3 Scikit-Learn Package

We use Scikit-Learn package to train the SVM classifiers.
4 Results

The results of this project are as follows:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Examples</th>
<th>Accuracy</th>
<th>Rank on Leaderboard</th>
<th>Rank in UMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>8160</td>
<td>0.78529</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Public Test</td>
<td>153</td>
<td>0.82353</td>
<td>6th</td>
<td>1st</td>
</tr>
<tr>
<td>Private Test</td>
<td>153</td>
<td>0.8170</td>
<td>5th</td>
<td>3rd</td>
</tr>
</tbody>
</table>

Here is the results of applying different scores for training:

<table>
<thead>
<tr>
<th>Scores</th>
<th>Training Accuracy</th>
<th>Public Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>QANTA+IR Wiki</td>
<td>0.78235</td>
<td>0.77778</td>
</tr>
<tr>
<td>QANTA+IR Wiki+BoW</td>
<td>0.78394</td>
<td>0.79085</td>
</tr>
<tr>
<td>QANTA+IR Wiki+BoW+TV</td>
<td>0.78529</td>
<td>0.82353</td>
</tr>
</tbody>
</table>

It turns out that the performance gets better when we apply more types of scores, which means that the cosine similarity of BoW and topic vector is helpful to improve our model.

5 Error Analysis

We analyse some common errors in our approach. First, all our approaches are based on the candidate guesses provided by QUANTA and IR Wiki. Our approaches are not able to handle question when the correct answer is not in the candidate list. We have 16 such kind of questions in the test. For those questions, 15 of them have only 1 sentence, another one has 2 sentences, which means they are difficult. 9 out of 16 are from category 'Lit', in which the content of the question may strongly align with books or authors. To tackle these problems, we would work on extracting key words from questions in the future. For 'Lit' category, words like 'man', 'woman', 'novel', 'story' could help us distinguish the intention of the questions. For other categories, we can extract some rare words in the sentence, such as 'elizabeth i' in question 3864, and search related candidates based on those queries.

Our result achieves 251 correct out of the 306 tests, comparing with the two baselines QUANTA (228/306) and IR Wiki (215/306). As shown in Fig. 1, our results assign reasonable scores for the candidate guesses, since the rank of the correct answers are high for our incorrect results (17 out of 39 hit the correct answer with the second best guess).

There are several reasons that may lead to errors. The questions with less sentences are more difficult for this problem, as the average sentence position for error question is 0.69, while it is 1.55, which is twice length as for all the tests. For the four categories, "Science" seems to be harder since the error rate is 9/42 comparing to the average 39/306. The specific science concept, such as "mitochondrion", seems harder than a general concept, such as "stomach". One of the reasons may be lack of training data for the specific topics. To utilize key words as we suggested may still help, for example, '1925', 'fermions', 'bosons' in question 147637.

We use average rank, which indicates where the correct answer appears according to the scores, to analyse the four features introduced in Sec. 2. The provide DNN and IR Wiki scores are quite powerful with average rank 0.75 and 0.74, our BoW and TV features are not that impressive (average rank 5.13 and 13.88). For the error results, IR Wiki is the good clue, which
has average rank 0.15 better than average 0.74 on all test. Meanwhile, there are 6 questions among the error questions do not contain the correct answer in the guesses of IR Wiki. Hence another direction is to generate powerful features and take advantage of the given features. Note that we also generate different results with the binary classification approach described in Sec.2.5. The sophisticated way to combine results would help a lot. To conclude, keywords extraction is our next step to nail the problem.

Figure 1: The histogram of the rank of the correct answer in our results. We set the maximum number to shown as 20. Rank -1 means the correct answer is not in the candidate list provided. Rank 0 means our method correctly find the answer. Even when our approach does not assign the largest score to the correct answer, we give the correct answer reasonable score since their rank is still high.

6 Relation with Class

In this project, we employ stemming and POS tagging (though doesn’t help) to preprocess question text and Wiki text. The former one is introduced in linguistic ambiguity section and the latter one is introduced in syntax section.

We also implement the topic model algorithm, LDA, on our own. It helps us to analyze the topic distribution of question and Wiki text.

References

