DeepIntell at the NTCIR-13 STC-2 Task

Chunyue Zhang
DeepIntell Co. Ltd, China
cyzhang@deepintell.cn

Dongdong Li
DeepIntell Co. Ltd, China
ddli@deepintell.cn

Huaxing Shi
DeepIntell Co. Ltd, China
shihuaxing@deepintell.cn

ABSTRACT

This paper provides an overview of DeepIntell's system participated in the Short Text Conversation (STC2) task of Chinese at NTCIR-13. Previous STC of NTCIR-12 is a conversation task which can be defined as an IR problem, i.e.,retrieval based a repository of post-comment pairs. STC2 of NTCIR-13 provided a transparent platform to compare the generation-based method and IR method via comprehensive evaluations. In this paper, we adopt the IR method and propose a SVM based ranking method with the deep matching features to score the post-comment pairs. Our framework is based on the following four steps: 1) preprocessing, 2) building search index, 3) comment candidates generation, 4) comment candidates ranking. Our best run achieves 0.5564 for mean P+,0.4323 for mean nDCG@1 and 0.5594 for mean nERR@10. The evaluation of submitted results empirically shows our framework is effective in all these terms.

Team Name

DeepIntell

Subtasks

Short Text Conversation-2 (Chinese)

Keywords

retrieval, deep matching, ranking, conversation

1. INTRODUCTION

With the emergence of social media and the wide spread of mobile devices, conversation via short texts has become an important way of communication. Short Text Conversation-2 (STC2) [7] is a NTCIR-13 task which tackles the following research goal: a STC2 system can reuse an old comment from the repository to satisfy the author of the new post, e.g. the *retrieval*-based method.

Given a new post, this task aims to retrieve an appropriate comment from a large post-comment repository. The retrieved comment is judged from four facets: Fluent, Coherent, Self-sufficient and Substantial.

In this paper, we propose a SVM ranking based method with the deep matching features to score the new post-comment pairs retrieved from the repository based on the following specific steps: preprocessing, building search index, comment candidates generation and comment candidates ranking. We then show the effectiveness of the method of measuring similarity between short texts.

Our contribution is twofold: 1) we apply Whoosh to index the repository. In this way, it is very easy and fast to find the related information from the repository; and 2) we put forward a SVM ranking approach for candidates ranking to find the most appropriate comments for a new post.

2. SYSTEM ARCHITECTURE

In this section, we will respectively elaborate on two components of our system. Figure 1 illustrates the framework of our system.

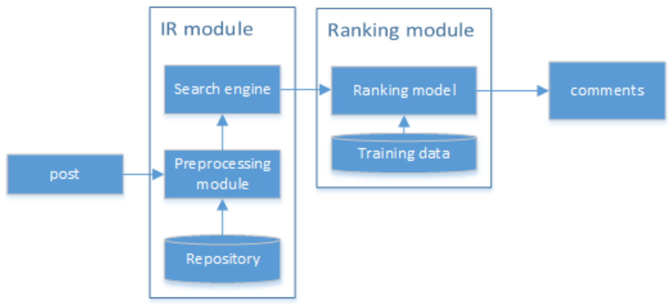

Figure 1: the DeepIntell STC2 system framework. It encapsulates two main components, i.e., information retrieval, semantic ranking module.

IR module is composed of preprocessing module, search engine and repository of post-comment pairs. We adapted the *Whoosh* [1] as the search engine, which use BM25F algorithm for scoring. The top N retrieved results should be reranked in following ranking module, to provide the final top 10 comments .

2.1 Preprocessing

In this module, we do data cleaning, word segmentation, removal of stop words. We convert traditional Chinese to simplified Chinese and remove the peculiar symbols and extra punctuations in order to clean the text. Chinese short texts are written without any symbols between characters, therefore we need to segment them before using queries or building search indexes. After measuring the accuracy, performance, and interface compatibility, we use jieba [2] as the

[1]https://bitbucket.org/mchaput/whoosh
[2]https://pypi.python.org/pypi/jieba/

Chinese segmenter. In order to improve the accuracy and performance of search engines, we use a stop words list (766 stop words, including punctuation, tone words and conjunctions) to filter the meaningless words and symbols. The following example in Table 1 shows the origin text, segmentation result and filtering result.

2.2 Search Engine

There are millions post-comments pairs in the repository, and it's necessary to retrieve the most appropriate comment candidates for a new post efficiently. Therefore, using a mature open source search engine framework can ensure both stability and computational performance. In fact, there are several open source retrieval frameworks can meet our needs, such as Lucence, elasticsearch, and we choose Whoosh to build index of posts and comments, because of its flexibility and the medium size of the repository. The Whoosh is a fast, full-text indexing and searching library implemented in pure Python. The Whoosh's functional modules can be easily extended or replaced to meet our needs. The index building mainly through the preprocessing module and the search engine, post-comment pairs will be stored to the search engine, and in the search processes through the same modules to retrieve comments. The preprocessing module adopts Jieba as word segmentation tool. In addition, when building the index, the stop words are filtered to ensure the performance of search engine.

2.3 Comment Candidates Generation

In practice we change the size of comment candidates to measure the impact. In this paper **V1** is the strategy with 120 comments, and **V3** is the strategy with 180 comments. Given a new post, to generate 120 or 180 comment candidates we combine the results of three ways from the repository, as following:

- The top 40 or 60 comments, which are retrieved from the repository by Whoosh, according similarity of corresponding post and new post.
- The top 40 or 60 comments, which are retrieved from the repository by Whoosh, according similarity of comment themselves and new post.
- The top 40 or 60 comments, which are retrieved from the repository by Whoosh, according multi similarity of corresponding post and comments themselves and new post.

To generate comments list in each strategy, we use BM25F [4] (default algorithm in Whoosh) algorithm in the search engine. BM25F is an extension of the BM25 ranking function adapted to score structured documents. It is well known that the probability of relevance of a document increases as the term frequency of a query term increases, and that this increase is non-linear [5]. For these reasons most modern ranking functions use an increasing saturation function to weight document terms that appear in the query. We use the following rank function.

$$\begin{aligned} BM25F_d &= \sum_{t \in q \cap d} \frac{tf(t,d)}{k_1 + tf(t,d)} \times idf(t) \\ tf(t,d) &= \sum_{c \in d} w_c \times tf_c(t,d) \end{aligned} \tag{1}$$

where c represents each field contained in the document d, wc is the weight or boost factor for each field in the document and $tf_c(t,d)$ is the field term frequency function of the term t in the field c. This simple modification of the classical tf—idf equation allow us to compute the score of the documents considering the structural information.

2.4 SVM based Ranking Model

For the retrieval based STC2 task, given a new post, the search engine firstly retrieves the candidate posts and the corresponding comments by the search strategy talked above. To find the most appropriate comments for the given post, our system use a directly linear SVM based ranking model [2] on the relevance score of comment candidates for a new post. In our system, we collect the following features to capture the relevance between the given post and the retrieved post-comment pairs.

- Recall and precision of 2-gram: Recall and precision of 2-gram matches for the given post and the candidate comments.
- The BLEU-1 evaluation metrics in Machine Translation for the given post and the candidate comment.
- Edit distance: Defined as the least steps to transform the post sentence to candidate comment sentence using insertion, deletion and substitution.
- Longest Common Sub-sequence(LCS): The sum of all the longest same sub-sequence length between post and candidate comment.
- Cosine similarity: The given post and the candidate comment are assumed to be two vectors in a |V|-dimensional vector space. |V| is the number of unique terms in the corpus. Calculate the cosine between of two vectors. Also calculate the cosine between the give post and the candidate post.
- Word embedding: we use the average of word vectors of the given post and comment to represent the sentence, and calculate their cosine.
- Deep Matching measure: a feature representing the semantic matching between the post and comment using the DL methods discussed the section 2.5.

2.5 Deep Matching features

In this section, we use a deep matching framework [3] to measure the semantic matching between the post and comment. We resort to a CNN structure built on the output of an LSTM [1] layer, in order to represent a more composite distributed representation of post and comment.

The structure of the deep matching in our work is similar to the one in [8], as shown in Figure 2. We share the same parameters between the post and comments sides. A LSTM layer generates the distributed representation of the whole sentence, and the CNN layer and max-pooling operation extract the most significant information of the sentence, then utilize cosine similarity to measure their semantic distance. Following the same ranking loss in the [8], we define the training objective as a hinge loss.

$$L = max\{0, M - sim(p, c_+) + sim(p, c_-)\} \tag{2}$$

Table 1: The preprocessing step

Short text ID	repos-post-2000235240
Origin Text	【2013 年 7 款将会改变我们生活的未来派设备】 (The 7 futuristic equipments of year 2013, which will change our life)
Segmentation Result	【 2013 年 7 款 将 会 改变 我们 生活 的 未来派 设备 】
Filtering Result	2013 年 7 款 会 改变 生活 未来派 设备

where c_+ is a ground truth comment, and c_- is an instance comment randomly chosen from the entire comments space, and M is the constance margin. In our work, we define $sim(p,c) = cosine(p,c)$. We treat any post with more than one ground truth as multiple training examples, each for one ground truth.

In the STC2 task, the chosen comment will be labelled as +1 at most if the comment is very similar with the post at the word level, so we penalize the string similarity as $sim(p,c) = cosine(p,c) - ovp(p,c)$. Here $ovp(p,c)$ is defined the ratio of the number of words in the comments appeared in the post.

Further, we try two ways to improve the loss function. First way is to encourage the chosen comment that is more semantic relevant with the post, and the other way is to penalize the very high similarity as follows.

- **Sim Credit**:

$$L = max\{0, M - sim(p,c_+) + sim(p,c_-)\}+ \lambda \times max\{0, \theta_{low} - cosine(p,c_+)\} \tag{3}$$

Here θ_{up} means a similarity lower bound constant. And λ is a regularized factor. In this way we hope the minimum semantic similarity between the post and the comment is greater than θ_{low}.

- **Sim Penalty**:

$$L = max\{0, M - sim(p,c_+) + sim(p,c_-)\}+ \lambda \times max\{0, \theta_{up}, cosine(p,c_+)\} \tag{4}$$

Here θ_{up} means a similarity upper bound constant. In this way we hope the maximum semantic similarity between the post and the comment is less than θ_{up}.

3. EXPERIMENTS

3.1 Data Set

In the STC2 task, there are 219,174 Weibo posts and the corresponding 4,305,706 comments. There are 4,433,949 post-comment pairs. So each post has 20 different comments on average, and one comment can be used to respond to multiple different posts.

There are 769 query posts and each of them have about 15 comment candidates in the labelled ranking validation data. There are 11,456 post-comment pairs with "suitable(2)", "neutral(1)", and "unsuitable(0)" labels.

"Suitable" means that the comment is clearly a suitable comment to the post, "neutral" means that the comment can be a comment to the post in a specific scenario, while "unsuitable" means it is not the two former cases.

100 posts are used for test. We submitted up to five runs to the task. In each run, a ranking list of ten comments for each test query is requested.

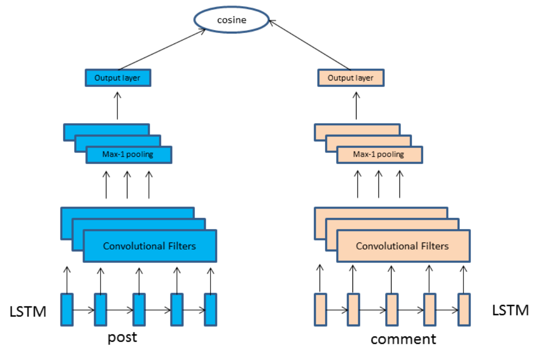

Figure 2: Deep Matching Architecture using CNN/LSTM

3.2 Evaluation Metrics

The evaluation metrics are nG@1, nERR@10 and P+[6].

nG@1 shows the quantity of effective result (such as L0, L1, L2 result) in the retrieved candidates. It will take three values: 0, 1/3 or 1 in this task.

nERR@10 shows the rank correctness of the candidates ranking, which means that the more effective result should be ranked as more front of the ranking list of retrieved candidates.

P+ depends most on the position of the best effective result in the ranking list of retrieved candidates. It gives the top ranked result the most ratio.

3.3 Experiments Results

First, we introduce all the hypermeters in our experiments here. We use the SVM^{rank} package [2] to train the rank model according the validation set. We just use the linear kernel and the c is set to 7000. For training the deep matching model, we use word2vec tookit to pre-train the $200d$ word embedding on the STC2 repository corpus. For the LSTM layer, we set the hidden vector is also 200. For the CNN layer, we set the filter size as [2,3], and the number of filters is 500. The batch size is 256 and the learning rate is 0.05. And we set the maximum epochs 10 to stop the training phase. We empirically set the θ_{low} as 0.3, and θ_{up} as 0.85. And we use the standard SGD to minimize the loss function.

For the test set, we submitted five runs for comparison and analysis:

- DeepIntell-C-R1: SVM ranking with word-level-feature and Deep Matching feature with **Sim Credit** loss on

Table 2: Ofcial STC2 results for team DeepIntell

	Mean nG@1	Mean P+	Mean nERR@10
DeepIntell-C-R1	0.4323	0.5564	0.5994
DeepIntell-C-R4	0.4077	0.5270	0.5774
DeepIntell-C-R2	0.3923	0.5258	0.5678
DeepIntell-C-R3	0.3773	0.5082	0.5484
DeepIntell-C-R5	0.3523	0.5023	0.5360

V1-retrieval results.

- DeepIntell-C-R4: SVM ranking with word-level-feature and Deep Matching feature on V1-retrieval results.
- DeepIntell-C-R2: SVM ranking with word-level-feature and Deep Matching feature with **Sim Credit** loss on V3-retrieval results.
- DeepIntell-C-R3: ranking just using Deep Matching scores with **Sim Credit** loss on V3-retrieval results.
- DeepIntell-C-R5: ranking just using Deep Matching scores with **Sim Penalty** loss on V3-retrieval results.

The official results of our system is shown in the Table 2. Compared DeepIntell-C-R1 with DeepIntell-C-R4, we can see the **Sim Credit** loss function improve the matching a lot. And compared with DeepIntell-C-R5, we can see the **Sim Penalty** doesn't work. Compared DeepIntell-C-R1 with DeepIntell-C-R3, we find the V1 retrieval strategy is more suitable for the STC2 task.

4. CONCLUSIONS

In this paper we proposed a system to rank candidate comments in the repository and find appropriate responses of a new post. Our best run achieves 0.5564 for mean P+, 0.4323 for mean nDCG@1 and 0.5594 for mean nERR@10 on the STC2 Chinese task.

5. REFERENCES

[1] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[2] T. Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

[3] Z. Lu and H. Li. A deep architecture for matching short texts. In *Advances in Neural Information Processing Systems*, pages 1367–1375, 2013.

[4] J. R. Pérez-Agüera, J. Arroyo, J. Greenberg, J. P. Iglesias, and V. Fresno. Using bm25f for semantic search. In *Proceedings of the 3rd international semantic search workshop*, page 2. ACM, 2010.

[5] S. Robertson, H. Zaragoza, and M. Taylor. Simple bm25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 42–49. ACM, 2004.

[6] T. Sakai, L. Shang, Z. Lu, and H. Li. Topic set size design with the evaluation measures for short text conversation. In *Asia Information Retrieval Symposium*, pages 319–331. Springer, 2015.

[7] L. Shang, T. Sakai, H. Li, R. Higashinaka, Y. Miyao, Y. Arase, and M. Nomoto. Overview of the NTCIR-13 short text conversation task. In *Proceedings of NTCIR-13*, 2017.

[8] M. Tan, B. Xiang, and B. Zhou. Lstm-based deep learning models for non-factoid answer selection. *CoRR*, abs/1511.04108, 2015.