The splab at the NTCIR-12 Short Text Conversation Task

Ke Wu[*], Xuan Liu, Kai Yu[*]

Key Laboratory of Shanghai Education Commission for Intelligent Interaction and Cognitive Engineering

SpeechLab, Department of Computer Science and Engineering

Shanghai Jiao Tong University, Shanghai, 200240, China

{ke.wu, 1097114522, kai.yu}@sjtu.edu.cn

ABSTRACT

The splab team participated in the Chinese subtask of the NTCIR-12 on Short Text Conversation Task. This task assumes that the existing comments in a post-comment repository can be reused as suitable responses to a new short text. Our task is to return 10 most appropriate comments to such a short text. In our system, we attempt to employ advanced IR methods and the recent deep learning techniques to tackle the problem. We develop a three-tier ranking framework to promote the most suitable comments in top position as much as possible. It consists of three components, i.e., search, lexical ranking and semantic ranking. In the search component, three different query generation methods are employed to boost the system's recall. In the lexical ranking, we exploit the training data of labelled post-comment pairs to score the comments in the candidate pool. In the final semantic ranking, we apply the deep learning techniques to convert the comment string or a short text string to a continuous, low-dimensional feature vector, re-score the final candidate comments and provide the 10 most reasonable comments to a short text. The evaluation of submitted results empirically shows our framework is effective in terms of mean nDCG@1, mean P+ and mean nERR@10.

Team Name

splab

Subtasks

Short Text Conversation (Chinese)

Keywords

short text conversation, keyword extraction, ranking SVM, CDSSM

1. INTRODUCTION

The Speech Lab of Shanghai Jiao Tong University participated in the NTCIR-12 Short Text Conversation Task [11]. This task aims to tackle a one-round dialogue problem via retrieval techniques given a large repository of short text conversation data. The task contains two subtasks, i.e., Chinese subtask and Japanese subtask. Our work focuses on the Chinese subtask.

[*]Corresponding authors.

Over the last decades, there is much effort on natural language conversation research, which models a complete dialogue session, typically with several rounds of interaction [16]. Short text conversation task is a simplified version of natural language conversation between human and machine. It simplifies natural multi-round dialogue to a single-round one formed by two short texts.

In the subtask, we build a single round of retrival-based dialogue system based on a repository of weibo data, which is provided by the competition's organizers. Too many irrelevant candidate comments can potentially hinder a system's ability to identify the appropriate response from the large pool of candidates [2] . Consequently, our system attempts to facilitate high short text conversation performance by a three-level ranking framework.

Our system consists of three componets: *search*, *lexical ranking* and *semantic ranking*. In the search component, we convert a short text (a query) to query terms and try to retrieve relevant contents as intial candidate comments. To boost the system efficiency, we use a threshold for the search hit list to narrow the candidate space to a small number of possiblities. However, such a brute-force approach may hamper the recall of the appropriate comments. In our system, we employ three different query terms conversions from an original short text, to generate three sets of candidates independently.

The lexical ranking component further identifies potential comments to a short text from the downstream candidates of the search component. In this component, we exploit a ranking model to score the downstream candidates and form a reduced candidate set.

In the semantic ranking component, we merge the three sets of reduced candidates from the lexical ranking component to form the input candidates, calculate the semantic similarities between the short text and these candidate comments based on deep learning models and generate the final plausible candidate list.

The remainder of this paper is organized as follows. Section 2 describes our systems in detail. Our experimental results are presented in Section 3 . We conclude in Section 4 .

2. OUR SYSTEM

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

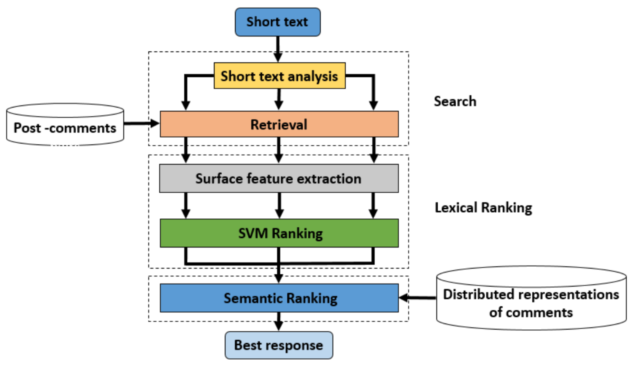

Figure 1: Our three-tier ranking framework. It encapsulates three main components, i.e., search, lexical ranking and semantic ranking.

2.1 Search Component

2.1.1 Short Text Analysis

Short text analysis is a key step for the search component. Typically, search query terms for a short text are generated by the short text analysis. In our system, we retain all possible segmented words and ignore all stopword terms [13] , which carry the negligible information, while converting a short text into query terms. We use all potential words with equal weights in multi-granularity word segmentation for the retrieval to improve search recall. In order to increase system performance, we have to determine the size of the search hit list to strike a balance between increased candidate recall and the processing time cost.

The query term generation strategy aforementioned usually provides an effective candidate list. However, the relevant comments may not be ranked in top positions because of a lot of uninformative query terms from all possible words of a short text in some cases. To avoid the adversity, we attempt to resort to the keyword identification to alleviate the noise information overload problem. For convenience, we use MG short for the above method.

Traditional keyword extraction techniques can help detect or recognize the focus with salient information, also called keywords for a short text. In our system, we use two state-of-the-art methods, i.e., $TFIDF$ and $TextRank$ [10] .

The idea of the $TFIDF$ method is to identify words that appear frequently in a short text but occur rarely in the entire background collection. Much work [3, 5, 15] has shown that TFIDF performs well in extracting keywords. The term frequency (TF) for a word t_i in a document is the number of times the word occurs in the document. The IDF value is:

$$IDF_i = \log(N/N_i) \tag{1}$$

where N_i denotes the number of the documents containing word t_i, and N is the total number of the documents in the collection.

In our system, a short text typically has a relatively small length. Consequently, IDF factor makes a major contribution while TF factor plays a minority role for keyword extraction of a short text. Therefore, we can consider that $TFIDF$ just employs a global information to detect keywords.

TextRank is another short text analysis technique used in our system. It employs graph-based methods similar to Google's PageRank algorithm [1]. In order to construct the text graph, TextRank considers words as vertices, defines co-occurrence relations within a window of maximum words as links between words or vertices and thus generates a potentially useful connections (edges). In addition, TextRank boosts its efficiency and accuracy by selecting only lexical units or words of a certain part of speech.

When the text graph is constructed, we iterate the following formula until convergence for the final scoring of words.

$$S(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} \frac{w_{ij}}{\sum\limits_{V_k \in Out(V_j)} w_{jk}} S(V_j) \tag{2}$$

where V_i is a word vertex, $S(V_i)$ denotes the score of V_i, w_{ij} is the strength of the connection between two vertices V_i , V_j, $In(V_i)$ is the set of word vertice that link to V_i, $Out(V_j)$ is the set of word vertice to which a vertex V_j points and d is a damping factor between 0 and 1, which serves as the role of modeling the probability of jumping from a given vertex to another random vertex in the text graph.

The performance of $TextRank$ depends on the constructed text graph and the mask POS set and hence we can consider TextRank identifies the salient words from the local text perspective.

In our system, the output of short text analysis module is the query terms with weights generated by MG, $TFIDF$ and $TextRank$, respectively.

2.1.2 Retrieval

Our system's retrieval module uses the open-source Lucene search engine [8] to index all post-comment pairs. We use the default similarity function in Lucene. The score between short text q and s is caculated by Eq. 3.

$$\text{score}(q,s) = \text{co}(q,s) \cdot \text{qn}(q) \cdot \sum_{t \in q} \{\text{tf}(t \in s) \cdot \text{idf}(t)^2 \cdot w_t \cdot \text{norm}(t,s)\} \tag{3}$$

where $\text{tf}(t \in s)$ is defined as the number of occurrences of the term t in the currently scored text s, $\text{idf}(t)$ stands for inverse document frequency of the term t, $\text{co}(q,s)$ denotes the proportional of the query terms are found in the specified text s by $\frac{|q \cap s|}{|q|}$, $\text{qn}(q)$ is a normalizing factor for comparable scores between texts, w_t is a weight of term t in the text q, $\text{norm}(t,s)$ encapsulates a few (indexing time) boost and length factors. Readers are referred to [8] for more details.

In this retrieval module, we retrieve the repository in both post and comment fields, select a certain amount of individual top results , for instance, 5K, and merge all possible comments as a candidate pool. Note that MG, $TFIDF$ and $TextRank$ are employed independently for the above procedure.

2.2 Lexical Ranking Component

The search component produces the intial comment candidate set on the basis of the text similarity. This component attempts to promote all the relevant comments to the top of a ranked list based on the downstream comment candidates by leveraging a small portion of labelled post-comment pairs.

Lexical ranking component contains two parts. One is surface feature extraction and the other is SVM ranking. The surface feature module collects and generates some features, such as similarity featues and matching features, which is fed into the SVM ranking model. The features used in the SVM ranking model are illustrated in Table 3. We use some search scores associated with a candidate comment for re-scoring the candidate set from the search component, in conjuction with additional matching features generated by considering character or word matching information [6, 14]. In our system, we also call these search scores associated with a candidate comment as similarity features. In addition, Post_Num is used as a feature which avoids a general response as much as possible. With all these features on hand, we wish to apply a well-trained SVM ranking model to improve our system's ability to rank the relevant comments in top position.

In the SVM ranking module, we use the ranking SVM [7] by Thorsten Joachims, which is an variant of the support vector machine algorithm. The ranking SVM exploits the techniques of SVM to learn a global ranking function F from an ordering O. In our system, we assume that F is a linear ranking function such that

$$\forall\{(\boldsymbol{x}_i,\boldsymbol{x}_j):y_i<y_j\in O\}:F(\boldsymbol{x}_i)>F(\boldsymbol{x}_j)\Leftrightarrow \boldsymbol{w}^T\boldsymbol{x}_i>\boldsymbol{w}^T\boldsymbol{x}_j \tag{4}$$

where $\boldsymbol{x}_i$ is an instance with label y_i and a weight vector $\boldsymbol{w}$ is learned by a learning algorithm.

The ranking SVM algorithm translates the above problem into an optimal problem as below.

$$\begin{aligned}\text{minimize:}\quad & L(\boldsymbol{w},\xi_{ij})=\frac{1}{2}\|\boldsymbol{w}\|^2+C\sum\xi_{ij}\\ \text{subject to:}\quad & \boldsymbol{w}^T\boldsymbol{x_i}\geq\boldsymbol{w}^T\boldsymbol{x_j}+1-\xi_{ij}\\ & \forall(i,j):\xi_{ij}\geq 0\\ & \forall\{(\boldsymbol{x}_i,\boldsymbol{x}_j):y_i<y_j\in O\}\end{aligned} \tag{5}$$

When ranking SVM is applied in our system, an instance (feature vector) is created from a labelled post-comment pair. The details of these features can be referred to Table 3. The instances from all labelled post-comment pairs are then combined in training. There is no difference in treatments toward the instances from different labelled post-comment pair. Furthermore, the well-trained ranking SVM model is used to score the downstream candidate comment set. Finally, the different search strategies along with the SVM ranking yield results that are aggregated for further processing.

2.3 Semantic Ranking Component

In this component, we recieve and re-rank the aggregated results of three different search strategies from the lexical ranking component by semantics. We attempt to generate a representation to capture fine-grained contextual structures for the similarity calculation of a short text and its candidate response.

Deep learning techniques have been proposed for natural language understanding [4, 12, 9]. In our system, we try to leverage deep learning techniques in our semantic ranking component and hence adapt the CDSSM [4, 12] to Chinese sentence embedding, which attempts to capture the rich contextual structures in a sentence or short text.

The architecture of our adopted Chinese CDSSM is illustrated in Figure 2. The model encampasses four layers. The first layer is word n-gram layer. In this layer, we obtain a word n-gram by moving a contextual sliding window

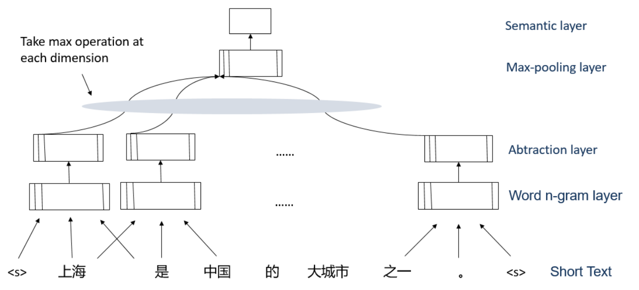

Figure 2: The simplified version of Chinese CDSSM. In order to obtain a full window for a word at any position in the input word sequence, we add a special padding word, <s>, at the beginning and the end of the input word sequence.

over the input word sequence (such as a short text or a response) similar to CDSSM and represent each word n-gram as the sum of the embeddings of the words it contains. The second layer, i.e., abstraction layer, generates a high-level feature vector for each word n-gram representation. The max-pooling layer discovers and catenates salient features from abstraction layer to form a fixed-length feature vector. In the semantic layer, we apply a regular feed-forward fully connected neural network to the fixed-length feature vector to obtain the final sentence embedding. In this model, we perform the non-linear transformation by Eq. 6.

$$h_t=\tanh(W\cdot h_{t-1}) \tag{6}$$

where W is the feature transformation matrix and t denotes the level of a hidden layer. tanh is used as the activation function of the neurons.

$$\tanh(x)=\frac{1-\exp(-2x)}{1+\exp(-2x)} \tag{7}$$

Note that, in our model, the feature transformation matrix from word n-gram layer to abstraction layer is shared among all word n-gram representations.

Given a short text and a set of comments to be ranked, we first use the aforementioned model to obtain the semantic vector representations for the short text and all candidate comments. Then we employ the cosine similarity between the short text and each candidate comment to measure the relevance score between them. We can formally define the semantic relevance score between a short text q and a candidate comment c as below.

$$R(q,c)=\text{cosine}(\boldsymbol{y}_q,\boldsymbol{y}_c)=\frac{\boldsymbol{y}_q^T\boldsymbol{y}_c}{\|\boldsymbol{y}_q\|\|\boldsymbol{y}_c\|} \tag{8}$$

where $\boldsymbol{y}_q$ and $\boldsymbol{y}_c$ are the semantic vectors of the short text q and the candidate comment c, respectively.

In order to determine the parameters of the model, we have to define the objective function. For short text conversation task, we wish to have higher relevance between a post and the corresponding comment than that between the post and other comments. Similar to [4], we convert the semantic relevance score between a short text and the corresponding comment to the posterior probability of the comment given the short text via softmax operation.

$$p(c\,|\,q)=\frac{\exp(\gamma R(q,c))}{\sum_{c'\in C}\exp(\gamma R(q,c'))} \tag{9}$$

Table 1: Features of re-ranking the candidate set from the search component

Types	Features	Meanings
Similarity Features	Q2C	Similarity between the query q and the candidate comment c
	Q2P_Ave	Average of the similarities between the query q and the posts with which the candidate comment c is paired
	Q2P_Max	Maximum of the similarities between the query q and the posts with which the candidate comment c is paired
	Q2P_Min	Minimum of the similarities between the query q and the posts with which the candidate comment c is paired
Matching Features	LCS	Length of the longest common string between the query q and the candidate comment c
	LCS_Rate	Ratio of LCS to the length of the candidate comment
	Co_Size	Number of co-occuring words between the query q and the candidate comment c
	Co_Rate	Ratio of Co_Size to the number of words in the candidate comment c
	Co_IDF_Sum	Sum of IDFs of co-occuring words between the query q and the candidate comment c
	Co_IDF_Ave	Average of IDFs of co-occuring words between the query q and the candidate comment c
Others	Post_Num	Number of the posts with which the candidate comment c is paired

where $(q,\, c)$ is a (post, comment) pair, C is the set of candidate comments to be ranked and γ is a smoothing factor in the softmax function.

In the training phase, the model parameters are learned to maximize the likelihood of the corresponding comment given a post across the post-comment repository. In other words, we can obtain the model parameters by minimizing the following loss function.

$$L(\boldsymbol{\theta}) = -\log \prod_{(q,c)} p(c \mid q) \qquad (10)$$

where $\boldsymbol{\theta}$ denotes the parameters of the model.

The similarity model is illustrated in Fig. 3. We can observe that the semantic tranformation structure of a post text has the same as that of a comment text. Two different approaches are adopted for the weight parameter tuning. One is that the weight update is asynchronous for post text and comment text and the other is that the weigh update is shared for a post text and a comment each time. In other words, we can consider the conversation of post text to its semantic feature vector is different from that of comment text while we also assume that the conversation to semantic feature vector is the same for both post text and comment text. In our experiments, we empirically found that the part-of-speech tagger information of a short text is useful for the generation of the semantic feature vector. Therefore, we use as our word-n-gram representation, the catenation of the original word n-gram feature and one-hot representation of POS taggers it contains for each word.

When we compute the relevance score between a short text and a candidate comment, we use three different strategies to transform them into the semantic feature vectors as shown in Table 2. The asynchronous weight update policy for model training indicates that the weight matrices are updated according to the objective function while the shared weight update policy implies that in Fig. 3, the left structure shares the same parameters with the right structure in training.

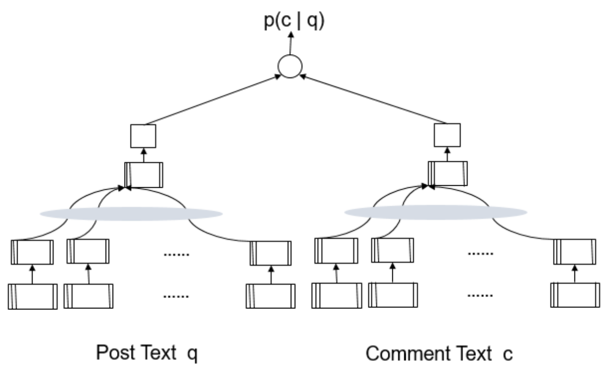

Figure 3: The similarity model Structure. The left of the circle has a common structure with the right of the circle. The structure in the left/right side denotes that of the post text q / the comment text c

Table 2: Three different transformation strategies from a text string to a feature vector in a continuous, low-dimensional space. L/R denotes that we use the left/right model structure in Fig. 3 to convert a text string to a feature vector.

No.	weight update	post vector	cmnt vector
1	asynchronous	L	R
2	asynchronous	R	R
3	shared	L or R	L or R

3. SUBMITTED RESULTS

In Chinese short text conversation task, we totally submit three runs. The evaluation results are listed in Table 3. splab-C-R1 uses the framework illustrated in Fig. 1. In the semantic ranking phase, it employs ranking SVM to combine three different semantic relevance scores. splab-C-R2 and splab-C-R3 just apply the semantic ranking component in Fig. 1. splab-C-R2 uses the first method in Table 2 to generate the feature vectors whereas splab-C-R3 employs the third method in Table 2 to obtain the feature vectors.

We observe in Table 3 that splab-C-R1 performs much better than splab-C-R2 and splab-C-R3 in terms of three metrics, i.e., Mean nDCG@1, Mean P+ and Mean nERR@10. To our surprise, both pure semantic ranking methods, i.e., splab-C-R2 and splab-C-R3 don't have a good performance. This may suggest that our semantic ranking is not good at bringing the suitable candiate comments to top positions in the overly large candidate pool with much noise while it can eliminate the irrelevant candiate comments from a small candidate set with few noise comments. Meanwhile, this observation also leaves room to improvement for the semantic ranking method.

4. CONCLUSIONS

Natural language conversation between human and computer is a very iinteresting and challeging research topic. In the last decades, there are a lot of effort on the research. However, there is little progress in this area. With the proliferation of the social media such as Twitter and Weibo, the large amount of short text conversation data are available online. This provides us a new opportunity to rethink and crack the problem.

In this task, we attempt to combine the some IR technologies with the deep learning techniques to attack the problem. Our system uses a three-tier ranking strategy to identify the potential responses to a short text. A basic idea in achieving high short text conversation performance is to collect sufficient candidate comments including the suitable responses in the small candidate pool.

In this paper, we described our system's three-pronged strategy for identifying proper responses that balance high candidate recall and processing time for candidate scoring. In the search component, we use three different methods to formulate effective queries to retrieve the satisfactory candidates as many as possible, respectively. In the lexical ranking component, we employ surface feature extraction module to extract some lexical features to re-rank the three individual downstream candidate pools. Then we choose top 10 candidate comments from each candidate set and aggregate them as the final candidate pool. In the semantic ranking component, we leverage the ranking SVM to combine three different transformation strategies from a text string to a continuous, low-dimensional feature vector for final fine-grained semantic ranking.

The evaluation on a test set of 100 test queries provided by the organizers shows that our three-tier ranking system is effective. Overall, all strategies make positive contributions.

Acknowledgement

This work was supported by the Program for Professor of Special Appointment (Eastern Scholar) at Shanghai Institutions of Higher Learning, the China NSFC project No. 61573241 and the Interdisciplinary Program (14JCZ03) of Shanghai Jiao Tong University in China.

5. REFERENCES

[1] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International Conference on World Wide Web 7*, WWW7, pages 107–117, 1998.

[2] J. Chu-Carroll, J. Fan, B. Boguraev, D. Carmel, D. Sheinwald, and C. Welty. Finding needles in the haystack: Search and candidate generation. *IBM Journal of Research and Development*, 56(3):6, 2012.

[3] E. Frank, G. W. Paynter, I. H. Witten, C. Gutwin, and C. G. Nevill-Manning. Domain-specific keyphrase extraction. In *Proceedings of IJCAI '99*, pages 668–673, 1999.

[4] J. Gao, P. Pantel, M. Gamon, X. He, and L. Deng. Modeling interestingness with deep neural networks. In *Proceedings of EMNLP*, pages 2–13, 2014.

[5] A. Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, EMNLP '03, pages 216–223, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[6] Z. Ji, Z. Lu, and H. Li. An information retrieval approach to short text conversation. *CoRR*, abs/1408.6988, 2014.

[7] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of KDD '02*, pages 133–142, 2002.

[8] M. McCandless, E. Hatcher, and O. Gospodnetic. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Manning Publications Co., Greenwich, CT, USA, 2010.

[9] G. Mesnil, X. He, L. Deng, and Y. Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Interspeech*, 2013.

[10] R. Mihalcea and P. Tarau. Textrank: Bringing order into texts. In *Proceedings of EMNLP 2004*, pages 404–411, Barcelona, Spain, July 2004.

[11] L. Shang, T. Sakai, Z. Lu, H. Li, R. Higashinaka, and Y. Miyao. Overview of the NTCIR-12 short text conversation task. In *Proceedings of NTCIR-12*, pages 1–100, 2016.

[12] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the*

Table 3: Evaluation results of Chinese short text conversation task for three runs

Run name	Mean nDCG@1	Mean P+	Mean nERR@10
splab-C-R1	0.2933	0.4735	0.4449
splab-C-R2	0.0967	0.2069	0.1831
splab-C-R3	0.0967	0.1896	0.1650

23rd International Conference on World Wide Web, pages 373–374. ACM, 2014.

[13] C. J. van Rijsbergen. *Information Retrieval, 2nd edition*. Butterworth-Heinemann, 1979.

[14] H. Wang, Z. Lu, H. Li, and E. Chen. A dataset for research on short-text conversations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 935–945, October 2013.

[15] A. M. Yaakov HaCohen-Kerner, Zuriel Gross. Automatic extraction and learning of keyphrases from scientific articles. In *Computational Linguistics and Intelligent Text Processing*, pages 657–669, 2005.

[16] S. Young, M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu. The hidden information state model: A practical framework for pomdp-based spoken dialogue management. *Comput. Speech Lang.*, 24(2):150–174, Apr. 2010.