Nders at NTCIR-13 Short Text Conversation 2 Task

Liansheng Lin[1], Han Ni[2], and Ge Xu[3]

[1] NetDragon Websoft Inc., Fuzhou, China
linliansheng@nd.com.cn
[2] NetDragon Websoft Inc., Fuzhou, China
nihan@nd.com.cn
[3] Minjiang University, Fuzhou, China
XuGeNLP@nd.com.cn

Abstract. This paper describes our approaches at NTCIR-13 short text conversation 2 (STC-2) task (Chinese). For a new post, our system firstly retrieves similar posts in the repository and gets their corresponding comments, and then finds the related comments directly from the repository. Moreover, we devise a pattern-idf to rerank the candidates from above. Our best run achieves 0.4780 for mean nG@1, 0.5497 for mean P+, and 0.5882 for mean nERR@10, and respectively rankes 4th, 5th, 5th among 22 teams.

Keywords: Short text conversation, LSA, LDA, Word2vec, LSTM, Pattern-idf

1 Introduction

We participated in the NTCIR-13 Short Text Conversation 2 (STC-2) Chinese subtask. Given a new post, this task aims to retrieve an appropriate comment from a large post-comment repository (Retrieval-based method) or generate a new appropriate comment (Generation-based method). Our system chooses the retrieval-based method.

The retrieved or generated comment for the new post is judged from four criteria: Coherent, Topically relevant, Non-repetitive and Context independent[1]. The primary criterion for a suitable comment we consider is topically relevant. In other words, this comment should be talking about the same topic with the given post. We train LSA[2] model and LDA[3] model to obtain the degree of topic relateness, Word2Vec[4] model to obtain the semantic similarity between new post and retrieved comment. By combining them together, we proposed a simialrity score to search comment candidates, and achieves a good performance.

Based on a hypothesis, similar posts has similar corresponding comments, we try to find the similar posts to the new post and get their corresponding comments as a supplement for the candidates. In addition to Word2Vec model and LSA model, we also introduce langugae model trained by LSTM[5][6][7] to compute post-post similarity.

In the last step, we rank the cadidates by Random Walk and pattern-idf respectively. Result shows that pattern-idf improves the performance while Random Walk deteriorates it instead.

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 make conclusions in Section 4 .

2 System Architecture

The architecture of our system is described as Figure 1. It includes the following components.

2.1 Preprocessing

There are some traditional Chinese in raw text which will cause incorrect word segmentation, so we convert traditional Chinese to simplified Chinese with nstools[4]. Moreover, we convert full-width characters into half-width ones.

Unlike English words in a sentence are separated by spaces, Chinese short texts are written without any symbol between characters. So the word segmentation becomes necessary. We choose nlpir[5] to segment the chinese text. After segmentation, our system filters meaningless words and symbols according to Chinese stop words list in order to clean the result.

The following example in Table 1 shows the raw text, segmentation result without traditional-simplified conversion(Segmentation Result 1), segmentation result with traditional-simplified conversion (Segmentation Result 2), and clean result.

Table 1. The preprocessing result

Short Text ID	test-post-10440
Raw Text	去到美國，还是吃中餐！宮保雞丁家的感覺～
Segmentation Result 1	去 到 美 國 , 还 是 吃 中餐 ! 宮 保 雞 丁 家 的 感 覺 ~
Segmentation Result 2	去 到 美国 , 还 是 吃 中餐 ! 宫保鸡丁 家 的 感觉 ~
Clean Result	去 到 美国 还 是 吃 中餐 宫保鸡丁 家 的 感觉

2.2 Similarity Features

In order to compute the degree of similarity or relateness between two sentences, we convert text sentence into continuous vector representations with some techniques including LSA, LDA, Word2Vec, LSTM.

[4] https://github.com/skydark/nstools
[5] http://ictclas.nlpir.org/

LSA Latent semantic analysis (LSA) is a technique of analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. LSA assumes that words that are close in meaning will occur in similar pieces of text (the distributional hypothesis). A matrix containing word counts per paragraph (rows represent unique words and columns represent each paragraph) is constructed from a large piece of text and a mathematical technique called singular value decomposition (SVD) is used to reduce the number of rows while preserving the similarity structure among columns.[2] We combine each post with its corresponding comments to be a document, then we train LSA model on these documents with gensim[6]Ḟrom trained model, we can get vector for each chinese word. Then, we can get vector representation of a sentence by Eq.1:

$$V = \frac{1}{n} * \sum_{1}^{n} v_i \tag{1}$$

Here, capital V refers to vector of a sentence, v refers to vector of each word in the sentence, and n is the length of the sentence.

LDA Latent Dirichlet allocation (LDA) is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's creation is attributable to one of the document's topics.[3] Like training LSA model, we combine each post with its corresponding comments to be a document, then we train LDA model on these documents with gensim. With trained LDA model, we can transform new, unseen documents into LDA topic distributions. We regard a sentence(a post, or a comment) as a document, convert it into plain bag-of-words count vector, and then index LDA model to obtain a vector representation of the sentence.

Word2Vec Word2Vec is an efficient tool for computing continuous distributed representations of words[4]. We train our Word2Vec model on provided training text corpus with skip-gram architecture where window size is 10, vector length is 300 and min count is 20 to remove infrequent words. Vector representation for each chinese word is directly obtained from trained model. Then, we can get vector representation of a sentence by Eq.1.

LSTM ...

Cosine Similarity After convert sentence into vectors, we compute similarity of two sentences by cosine similarity:

$$Sim(s_1, s_2) = \frac{V_1 * V_2}{\|V_1\| \, \|V_2\|} \tag{2}$$

[6] A python library for topic modelling, https://radimrehurek.com/gensim/index.html

Here, V_1 refers to vector representation of sentence s_1, V_2 refers to vector representation of sentence s_2.

With sentence vectors from different models, we get corresponding sentence similarity. We use Sim_{LSA} to denote sentence similarity based on LSA model, analogously, Sim_{LDA} to denote sentence similarity based on LDA model and Sim_{W2V} to denote sentence similarity based on Word2Vec model.

2.3 Candidates Generation

Similar Posts Based on a hypothesis that similar posts has similar corresponding comments, we firstly find top-10 similar posts with a ranking score combining LSA, Word2Vec, and LSTM model:

$$Score_{q,p}(q,p) = Sim_{LSA}(q,p) * Sim_{W2V}(q,p) * Sim_{LSTM}(q,p) \tag{3}$$

Here, q denotes the query(the new post) p denote the post from repository.

Then, we get corresponding comments to the top-10 similar posts as first comment candidates, denoted as C_1.

Comment Candidates Since Word2Vec model captures semantic similarity, LSA or LDA reflects topic relateness, we combine LSA or LDA with Word2Vec respectively to directly retrieve top-N appropriate comments to the new post from all comments in the repository. N is equal to the number of comment candidates C_1.

$$Score^1_{q,c}(q,c) = Sim_{LSA}(q,c) * Sim_{W2V}(q,c) \tag{4}$$

$$Score^2_{q,c}(q,c) = Sim_{LDA}(q,c) * Sim_{W2V}(q,c) \tag{5}$$

Here, q denotes the query(the new post), c denotes the comment from repository.

We combine the retrived top-N comments by Word2Vec model and LSA(or LDA) model with C_1 as final comment candidats for further ranking.

2.4 Ranking

TextRank TextRank[8] is a graph-bsed ranking model for text processing. Graph-based ranking algorithms are essentially a way of deciding the importance of a vertex within a graph, based on global information recursively drawn from the entire graph. The basic idea implemented by a graph-based ranking model is that of "voting" or "recommendation". When one vertex links to another one, it is basically casting a vote for that other vertex. The higher the number of votes that are cast for a vertex, the higher the importance of the vertex. Moreover, the importance of the vertex casting the vote determines how important the vote itself is, and this information is also taken into account by the ranking model. Hence, the score associated with a vertex is determined based on the votes that are cast for it, and the score of the vertices casting these votes.

We consider an undirected weighted TextRank algorithm in our system. Formally, let $G = (V, E)$ be a undirected graph with the set of vertices V and and set of edges E, where E is a subset of $V \times V$. For a given V_i, let $link(V_i)$ be the set of vertices that linked with it. The score of a vertex V_i is define as follow:

$$WS(V_i) = (1 - d) + d * \sum_{j \in link(V_i)} w_{ij} * WS(V_j) \tag{6}$$

We add each unique word in candidates as a vertex in the graph and use a co-occurrence relation as edges between vertices in the graph. The edge is weighted by word2vec similarity between two words and the number of their co-occurrence. Here co-occurrence means two words co-occur within a window of maximum W words, where the window size W is set to be 5 in our system.

In our system, the TextRank value for each vertex refers to the importance of the word in candidates. We compute the TextRank value iteratively.

Firstly, for comment candidates, we create a dictionary D - a mapping between words and their integer ids. Such that, each unique word is mapped to a integer range from 0 to $k-1$, k is the size of the dictionary. We use D_i to denote a word whose id is i.

Therefore, the w_{ij} is defined as

$$w_{ij} = cnt * Sim(D_i, D_j) \tag{7}$$

When we scan the candidates sentence by sentence, if the word D_i and D_j co-occur within the window, the count for them increases by 1. The cnt in Eq.7 refers to the total count after scanning.

Then, we construct a $k \times k$ matrix $\boldsymbol{M}$, defined as

$$M_{ij} = \begin{cases} w_{ij} & j \in link(D_i) \\ 0 & otherwise \end{cases} \tag{8}$$

At time t = 0, We initiate a k-dimension vector $\boldsymbol{R}$, where each entry is defined as the inverse document frequency (idf) of the word:

$$R_i = idf(D_i) \tag{9}$$

At each time step, the computation yields:

$$\boldsymbol{R(t+1)} = d\boldsymbol{M}\boldsymbol{R(t)} + \frac{1-d}{k}\boldsymbol{1} \tag{10}$$

The computation ends when for some small ϵ, $|\boldsymbol{R(t+1)} - \boldsymbol{R(t)} < \epsilon|$, Where we set $\epsilon = 0.000,000,1$.

Since we get the score $R(D_i)$ for each word D_i, the score for each comment candidate c is calculated as:

$$Rank_{TextRank}(c) = \frac{\sum_{D_i \in c} R(D_i)}{len(c)} \tag{11}$$

Where, $len(c)$ refers to the number of words for comment c.

Finally, we use $Rank_{TextRank}$ to rank the comment candidates and get top-10 comments as our Nders-C-R3 results for each given new post.

Pattern-Idf Consider the hypothesis, similar post has similar corresponding comments. In other words, for the corresponding comments of similar post, the word distribution may also be similar. Therefore, we can use word distribution of the corresponding comments of post in the repository to infer that of the new post.

The conditional probability of word D_i in comment given word D_j in post is defined as:

$$Prob(D_i|D_j) = \frac{count(D_i, D_j)}{count(D_j)} \tag{12}$$

Then, we define Pattern-Idf as:

$$PatternIdf(D_i|D_j) = idf(D_j) * \log Prob(D_i|D_j) \tag{13}$$

For each comment c in candidates, given a query (new post) q, we calculate the score by Pattern-Idf as follow:

$$Score_{PatternIdf}(q, c) = \frac{\sum_{D_j \in q} \sum_{D_i \in c} PatternIdf(D_i|D_j)}{len(c) * len(q)} \tag{14}$$

Then we define rank score as follow:

$$Rank_{PatternIdf} = (1 + Score_{PatternIdf}(q, c)) * Sim_{w2v}(q, c) * Sim_{LDA}(q, c) \tag{15}$$

Finally, we use $Rank_{PatternIdf}$ to rank the comment candidates and get top-10 comments as our Nders-C-R2 results for each given new post.

TextRank + Pattern-Idf In this method, We add each comment sentence in candidates as a vertex in the graph and use Word2Vec similarity as edges between vertices in the graph.

At time t = 0, We initiate a l-dimension vector $\boldsymbol{P}$, here l is the number of comment candidates. And each entry of $\boldsymbol{P}$ is defined as the score of Pattern-Idf between the query (new post) q and corresponding comment c_i in candidates:

$$P_i = Score_{PatternIdf}(q, c_i) \tag{16}$$

Then, we construct a $l \times l$ matrix $\boldsymbol{M}$, defined as

$$M_{ij} = Sim_{W2V}(c_i, c_j) \tag{17}$$

At each time step, the computation yields:

$$\boldsymbol{P(t+1)} = d\boldsymbol{M}\boldsymbol{P(t)} + \frac{1-d}{l}\boldsymbol{1} \tag{18}$$

The computation ends when for some small ϵ, $|\boldsymbol{P(t+1)} - \boldsymbol{P(t)} < \epsilon|$, Where we set $\epsilon = 0.000,000,1$.

Finally, we get the score P_i for each comment in candidates. After sorting, the top-10 comments are obtained as our Nders-C-R2 results.

3 Experiments

3.1 Data Set

Table 2 shows the statistics of the retrieval repository, training data and test data. 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 in training data, each of which has about 15 candidate comments. Totally, there are 11,535 comments labeled with "suitable", "neutral", and "unsuitable". "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 query posts are used for test. Each team is permitted to submit five runs to the task. In each run, a ranking list of ten comments for each test query is requested.

Table 2. Statistics of dataset for Chinese subtask

Repository	#posts	219,174
	#comments	4,305,706
	#original pairs	4,433,949
Training Data	#posts	769
	#comments	11,535
	#labeled pairs	11,535
Test Data	#query posts	100

3.2 Evaluation Measures

Following the NTCIR-12 STC-1 Chinese subtask, three evaluation measures are used: nG@1 (normalised gain at cut-off 1), P+, and nERR@10 (normalised expected reciprocal rank at cutoff 10)[1].

nG@1 shows the quantity of effective result in the 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.

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.

3.3 Experimental Results

We submitted five runs for comparison and analysis:

1. Nders-C-R5: Use $Sim_{w2v} * Sim_{LDA}$ as a ranking score to directly retrieve and get top-10 comments from all comments.
2. Nders-C-R4: Use $Sim_{w2v} * Sim_{LSI}$ as a ranking score to directly retrieve and get top-10 comments from all comments.
3. Nders-C-R3: Use graph-based algorithm TextRank with words as vertices in the graph, and use score $Rank_{TextRank}$ to rank comment in the candidates and get top-10 comments.
4. Nders-C-R2: Use $Rank_{PatternIdf}$ as a ranking score to get top-10 comments from comment candidates.
5. Nders-C-R1: Use graph-based algorithm TextRank with comments as vertices in the graph and Pattern-Idf as initiate score for each comment to rank the comment in the candidates and get top-10 comments.

The official results of our five runs are shown in Table 3

Table 3. The official results of five runs for Nders team

Run	Mean nG@1	Mean P+	Mean nERR@10
Nders-C-R1	0.4593	0.5394	0.5805
Nders-C-R2	0.4743	**0.5497**	**0.5882**
Nders-C-R3	0.4647	0.5317	0.5768
Nders-C-R4	**0.4780**	0.5338	0.5809
Nders-C-R5	0.4550	0.5495	0.5868

Table 3 shows that, with the use of Word2Vec and LSA model, R4 achieves best result in our five runs for Mean nG@1, which finally ranks 4th among 22 teams.

The best results in our runs for Mean P+ and Mean nERR@10 are both R2, which introduces Pattern-Idf to rank the comment candidates generated by Word2Vec and LSA model(R4). The result of R2 improves against R4 and ranks 5th both for Mean P+ and Mean nERR@10 among 22 teams, which proves the effectiveness of the Pattern-Idf we devised.

However, the results of R3 are worse than that of R4 for all three metrics, which shows TextRank is not helpful for candidates ranking in this task.

Compare the result of R4 and R5, we find that for Mean P+ and Mean nERR@10 the result of R5 is better than that of R4, which shows LDA model outperform LSI model in this task.

4 Conclusions

In this paper, we propose an approach for STC-2 task of NTCIR-13. The LSA, Word2Vec and LSTM model are used to find similar posts. The LSA, LDA and Word2Vec model are used to retrieve comment candidates. A graph-based algorithm TextRank and the Pattern-Idf we devised are applied to rank the candidates. Results show that the Pattern-Idf we devised is effective for ranking

while TextRank not, and LDA model outperforms LSI model in retrieving candidates. Finally, our best run achieves 0.4780(R4) for mean nG@1, 0.5497(R2) for mean P+, and 0.5882(R2) for mean nERR@10, which respectively rankes 4th, 5th, 5th among 22 teams.

References

1. Lifeng Shang, Tetsuya Sakai, Zhengdong Lu, Hang Li, Ryuichiro Higashinaka, Yusuke Miyao. Overview of the NTCIR-12 Short Text Conversation Task, 2016.
2. Susan T. Dumais (2005). "Latent Semantic Analysis". Annual Review of Information Science and Technology. 38: 188–230.
3. Blei, David M, A. Y. Ng, and M. I. Jordan. "Latent dirichlet allocation." Journal of Machine Learning Research 3(2003):993-1022.
4. Mikolov, Tomas, et al. "Efficient Estimation of Word Representations in Vector Space." Computer Science (2013).
5. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.
6. Sundermeyer, Martin, R. Schlüter, and H. Ney. "LSTM Neural Networks for Language Modeling." Interspeech 2012:601-608.
7. Graves, Alex. "Generating Sequences With Recurrent Neural Networks." Computer Science (2013).
8. Mihalcea, Rada, and P. Tarau. "TextRank: Bringing Order into Texts." Unt Scholarly Works (2004):404-411.