Nders at NTCIR-13 Short Text Conversation 2 Task

Han Ni
NetDragon Websoft Inc., China
nihan@nd.com.cn

Liansheng Lin
NetDragon Websoft Inc., China
Fuzhou University, China
linliansheng@nd.com.cn

Ge Xu
Minjiang University, China
XuGeNLP@nd.com.cn

ABSTRACT

This paper describes our retrieval-based 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 two new methods. 1) LSTM-Sen2Vec model to get the vector of sentence. 2) 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.

Team Name

Nders

Subtasks

Short Text Conversation 2 (Chinese)

Keywords

Short Text Conversation, LSA, LDA, Word2Vec, LSTM, Pattern-IDF, Sen2Vec

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][2]. 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[3] model and LDA[4] model to obtain the degree of topic relateness, Word2Vec[5] 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 Sen2Vec model trained by LSTM[8][9][10] to compute similarity between two sentences.

In the last step, we rank the candidates by TextRank and Pattern-IDF. Results show that Pattern-IDF improves the performance while TextRank 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.

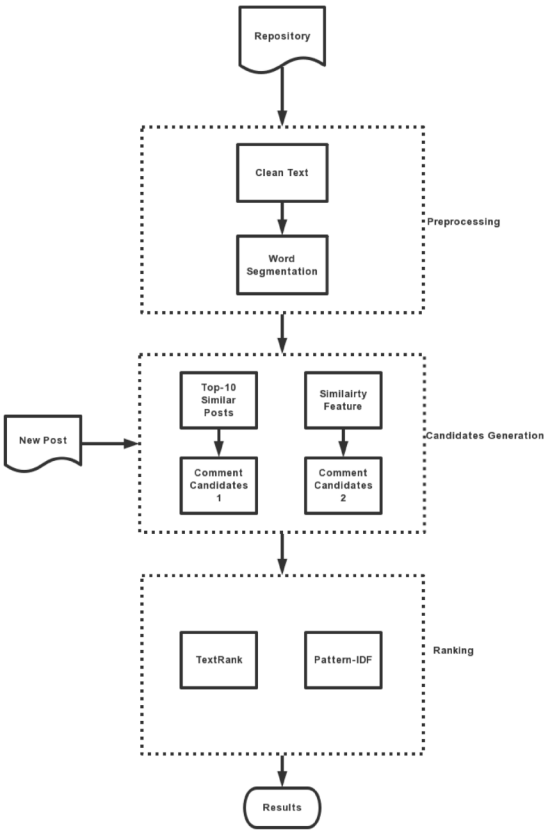

Figure 1: System Architecture

2. SYSTEM ARCHITECTURE

The architecture of our system is described as Figure 1.

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[1]. Moreover, we convert full-width characters into half-width ones. In addition, we replace all the number, datetime, url with token <_NUM>, <_TIME>, <_URL> respectively.

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[2] 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 and with traditional-simplified conversion (T-S conversion for short), and clean result.

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-Sen2Vec. While not using TF-IDF as the similarity feature directly, it would participate in training LDA, LSA models and calculate the similarity score by using Word2Vec, see 2.2.5.

2.2.1 TF-IDF

In information retrieval, TF-IDF, short for term frequency – inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval, text mining, and user modeling.

TF-IDF is the product of two statistics, term frequency and inverse document frequency. Various ways for determining the exact values of both statistics exist.

In the case of the term frequency $TF(t, d)$, the simplest choice is to use the raw count of a term in a document, i.e. the number of times that term t occurs in document d.

The inverse document frequency is a measure of how much information the word provides, that is, whether the term is common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient.

$$IDF(t, D) = \log \frac{N}{|d \in D : t \in d|} \tag{1}$$

Where, N refers to the total number of documents in the corpus, $N = |D|$. $|d \in D : t \in d|$ refers to the number of documents where the term t appears. If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the denominator to $1 + |d \in D : t \in d|$.

Then, TF-IDF is calculated as

$$TF - IDF(t, d, D) = TF(t, d) \cdot IDF(t, D) \tag{2}$$

2.2.2 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.[3] We combine each post with its corresponding comments to be a document, then we train LSA model (200 topics) on these documents with gensim[3]. From trained model, we can get vector for each chinese word. Then, we can get vector representation of a sentence by Eq.3:

$$V = \frac{1}{n} \cdot \sum_{i=1}^{n} v_i \tag{3}$$

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

2.2.3 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.[4] Like training LSA model, we combine each post with its corresponding comments to be a document, then we train LDA model (200 topics) 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.

2.2.4 Cosine Similarity

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

$$Sim(s_1, s_2) = \frac{V_1 \cdot V_2}{\|V_1\| \, \|V_2\|} \tag{4}$$

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, and Sim_{LDA} to denote sentence similarity based on LDA model.

2.2.5 Word2Vec

Word2Vec is an efficient tool for computing continuous distributed representations of words[5]. We train our Word2Vec model on provided post-comment pairs in repository and training data with skip-gram architecture where window size is 7, vector length is 300 and min count is 5 to remove infrequent words. Vector representation for each chinese word is directly obtained from trained model.

[1] https://github.com/skydark/nstools

[2] http://ictclas.nlpir.org/

[3] https://radimrehurek.com/gensim/index.html

Table 1: The preprocessing result

Short Text ID	test-post-10440
Raw Text	去到美國，还是吃中餐！宫保雞丁家的感覺～ Go to the USA, still eat Chinese food, Kung Pao Chicken, feeling like at home
Without T-S Conversion	去 到 美 國 , 还 是 吃 中餐 ! 宫 保 雞 丁 家 的 感 覺 ~
With T-S Conversion	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

Moreover, when using Word2Vec to calculate the sentence vector, we normalize every word's norm to its corresponding square root of value of IDF. Sum all words within a sentence and divede by its length, as below:

$$V = \frac{1}{n} \cdot \sum_{i=1}^{n} v_i \cdot \frac{\sqrt{IDF(w_i)}}{|v_i|} \tag{5}$$

Here, capital V refers to vector of a sentence, v_i refers to vector of each word w_i in the sentence, $IDF(w_i)$ refers to inverse document frequence for word w_i, and n is the length of the sentence.

Then, we calculate Word2Vec similarity by Eq.4, denoted as Sim_{W2V}.

2.2.6 *LSTM-Sen2Vec*

Word2Vec, LDA and LSA models capture the word meaning by the word distribution of its context, which ignores the order of words in a sentence. To take the order into account, we design a new model which can calculate the sentence vector with the sequence information, we call it LSTM-Sen2Vec.

Long short-term memory (LSTM) is a recurrent neural network (RNN) architecture that remembers values over arbitrary intervals. Figure 2 shows the architecture of LSTM. The C_t represents the cell state at time t and it is stored as a vector, which can theoretically remember all the previous information. The h_t represents current hidden state, which is also a output value at time t. The following equations describe the details of the architecture:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{6}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{7}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{8}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{9}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{10}$$

$$h_t = o_t * tanh(C_t) \tag{11}$$

Where $W_f, W_i, W_c, W_o, b_f, b_i, b_c, b_o$ refer to corresponding weights and bias.

Mikolov[6] and Zaremba et al.[7] use LSTM to predict the next word given the previous words as input and achieve a good perplexity. Inspired by their architecture of LSTM, we can use the vector of the final cell state to represent a sentence by feeding its words as input in sequence.

We train the model using unidirectional LSTM, whose architecture is shown as Figure 3.

Before training the LSTM model, we train the Word2Vec model as its word embedding vector and freeze it while the whole training process. The hidden size of LSTM cell is 300 which is the same as Word2Vec's dimension, and the total layers are 3, the forget bias is set to 1.0, with SGD as the optimizer and 0.95 decay every 2 epochs.

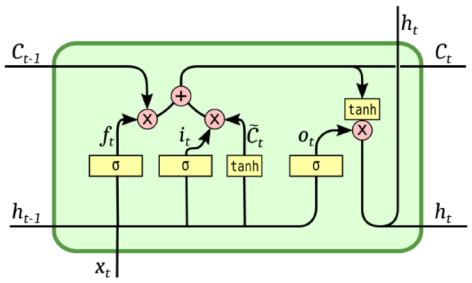

Figure 2: The repeating module in an LSTM contains four interacting layers.

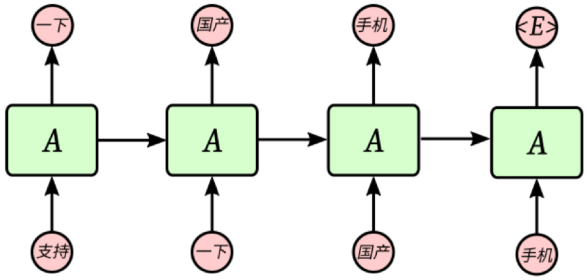

Figure 3: The unidirectional LSTM
We append an end token <*E*> in the end of word sequence to fix the length of input and output.
"支持一下国产手机" means: support the domestic mobile phone.

For a new post, we calculate its similarity with other post by the value of cosine similarity. When observe the result, we find that in the top-N similar posts, the last few words are extremely similar. Which suggests that the unidirectional LSTM model may assign much more weight to the end of a sentence.

To overcome the unbalance, we replace unidirectional LSTM with bidirectional LSTM. The traditional bidirectional LSTM architecture is shown as Figure 4. However, it doesn't make sense if we concatenate the outputs of forward and backward at the same time step , because at each time step both of them predict different words given the same word (such as "国产" concatenated with "支持", shown as Figure 4).

So we modify to make the concatenation rational, shown as Figure 5. Then we concatenate the outputs of both direc-

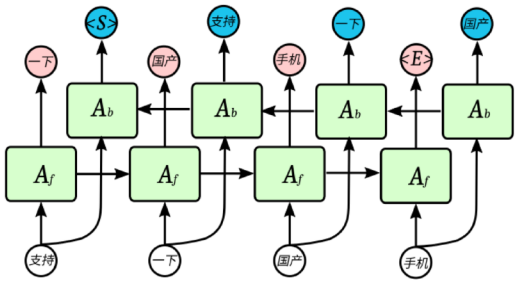

Figure 4: The Traditional Bidirectional LSTM
"支持一下国产手机" means: support the domestic mobile phone.

Table 2: Word-level perplexity on the Penn Tree Bank Dataset

	Validation set	Test set
Unidirectional LSTM	71.9	68.7
Modified Bidirectional LSTM	68.4	66.9

tions at each time step, which is 600-dimension. Through a full connection layer, it is reduced to dimension 300. We calculate the Euclidean distance of each output and its vector of Word2Vec model as the cost of neural networks. The word-level perplexity of our model on the Penn Tree Bank Dataset is shown as Table 2.

Finally, we concatenate the final cell states of two direction as the vector of the input sentence which is 600-dimension.

2.3 Candidates Generation

2.3.1 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, LDA, Word2Vec, and LSTM model:

$$Score^1_{q,p}(q,p) = Sim_{LDA}(q,p) * Sim_{W2V}(q,p) * Sim_{LSTM}(q,p) \quad (12)$$

$$Score^2_{q,p}(q,p) = Sim_{LSA}(q,p) * Sim_{W2V}(q,p) * Sim_{LSTM}(q,p) \quad (13)$$

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 by Eq.12 as first comment candidates, denoted as C_1, and corresponding comments by Eq.13 as second comment candidates, denoted as C_2.

2.3.2 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 or C_2.

$$Score^1_{q,c}(q,c) = Sim_{LDA}(q,c) * Sim_{W2V}(q,c) \quad (14)$$

$$Score^2_{q,c}(q,c) = Sim_{LSA}(q,c) * Sim_{W2V}(q,c) \quad (15)$$

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

We combine the retrived top-N comments by Eq.14 with candidates C_1 as our first final candidates. Then we rank them with $Score^1_{q,c}(q,c)$ and get top-10 comments as our results of Nders-C-R5.

We combine the retrived top-N comments by Eq.15 with candidates C_2 as our second final candidates. Then we rank them with $Score^2_{q,c}(q,c)$ and get top-10 comments as our results of Nders-C-R4.

Our results of R3, R2, R1 are based on the second final candidates from R4.

2.4 Ranking

2.4.1 TextRank

TextRank[11] is a graph-based 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:

$$WS(V_i) = (1-d) + d * \sum_{j \in link(V_i)} w_{ij} * WS(V_j) \quad (16)$$

where d is a damping factor[12] that is usually set to 0.85.

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

$$w_{ij} = cnt * Sim(D_i, D_j) \quad (17)$$

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.17 refers to the total count after scanning.

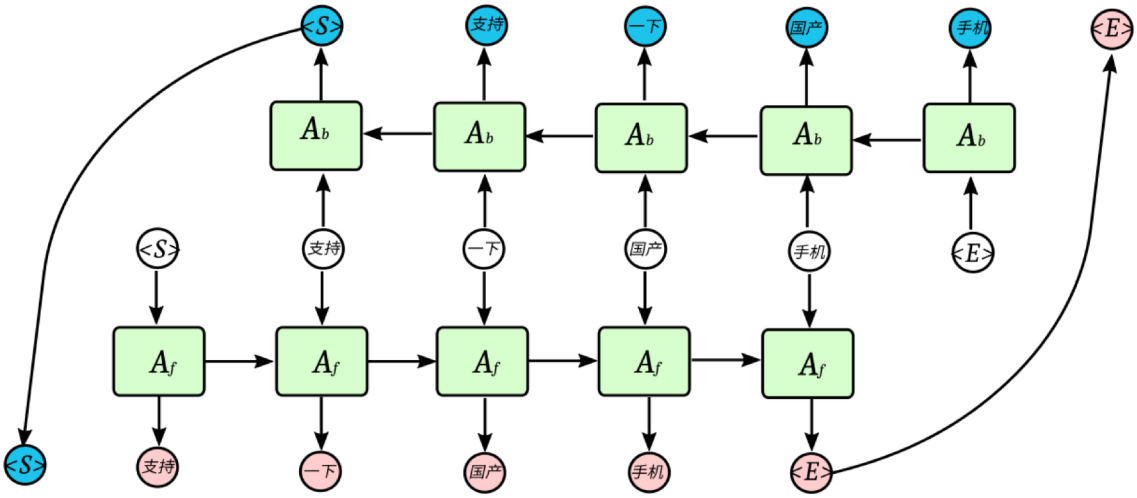

Figure 5: The Modified Bidirectional LSTM

In order to align the outputs of forward and backward direction, we append the first output value (indicates start token <*S*>) of backward direction to the outputs of forward, and append the last output value (indicates end token <*E*>) of forward direction to the outputs of backward."支持一下国产手机" means: support the domestic mobile phone.

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

$$M_{ij} = \begin{cases} w_{ij} & j \in link(D_i) \\ 0 & otherwise \end{cases} \tag{18}$$

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:

$$R_i = IDF(D_i) \tag{19}$$

At each time step, the computation yields:

$$\boldsymbol{R(t+1)} = d\boldsymbol{M}\boldsymbol{R(t)} + \frac{1-d}{k}\boldsymbol{1} \tag{20}$$

The computation ends when for some small ϵ, $|\boldsymbol{R(t+1)} - \boldsymbol{R(t)} < \epsilon|$, Where we set $\epsilon = 10^{-7}$.

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

$$Rank_{TextRank}(c) = \frac{\sum_{D_i \in c} R(D_i)}{len(c)} \tag{21}$$

where, $len(c)$ refers to the number of words in 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.

2.4.2 Pattern-IDF

Consider the hypothesis, similar posts have similar corresponding comments. In other words, for the corresponding comments of similar posts, 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. We present a new model, Pattern-IDF (PI).

Table 3: The example of Pattern-IDF

D_j	D_i	PI
中国移动 (China Mobile)	接通 (connect)	0.071725
中国移动	cmcc	0.067261
中国移动	资费 (Charges)	0.062408
中国移动	营业厅 (business Hall)	0.059949
中国移动	漫游 (roam)	0.059234
...	...	...
中国移动	我 (me)	0.028889
中国移动	是 (be)	0.027642
中国移动	的 (of)	0.026346

For word D_i in corresponding comment given word D_j in the post, we define PI as:

$$PI(D_i|D_j) = 1/\log_2 \frac{count_c(D_i) * count_p(D_j)}{count_{pair}(D_i, D_j) + 1} \tag{22}$$

Here $count_c$ refers to the number of occurrence in comments, $count_p$ in posts, $count_{pair}$ in post-comment pair.

Then, we normalize PI with dividing it by the summation of PI for all words in corresponding comment given word D_j in the post,

$$PI_{norm}(D_i|D_j) = \frac{PI(D_i|D_j)}{\sum_{k=1}^{n} PI(D_k|D_j)} \tag{23}$$

For convenience, all the PI below refers to PI_{norm}. Table 2 shows some example of Pattern-IDF.

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

$$Score_{PI}(q, c) = \frac{\sum_{D_j \in q} \sum_{D_i \in c} PI(D_i|D_j)}{len(c) * len(q)} \tag{24}$$

Then we define rank score as follow:

$$Rank_{PI} = (1 + Score_{PI}(q, c)) * Sim_{W2V}(q, c) * Sim_{LSA}(q, c) \quad (25)$$

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

2.4.3 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:

$$P_i = Score_{PI}(q, c_i) \quad (26)$$

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

$$M_{ij} = Sim_{W2V}(c_i, c_j) \quad (27)$$

At each time step, the computation yields:

$$\boldsymbol{P(t+1)} = d\boldsymbol{MP(t)} + \frac{1-d}{l}\boldsymbol{1} \quad (28)$$

The computation ends when for some small ϵ, $|\boldsymbol{P(t+1)} - \boldsymbol{P(t)} < \epsilon|$, Where we set $\epsilon = 10^{-7}$.

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

The repository consist of 219,174 Weibo posts and the corresponding 4,305,706 comments. There are 4,433,949 post-comment pairs in total. 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.

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 cut-off 10)[1][2].

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.

Table 4: 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

3.3 Experimental Results

We submitted five runs for comparison and analysis:

1. Nders-C-R5: Use LDA, Word2Vec and LSTM-Sen2Vec to retrieve similar posts and get corresponding comments, LDA and Word2Vec to retrieve appropriate comments from all comments, combine and rank them with $Score^1_{q,c}(q, c)$ and get top-10 comments as results.

2. Nders-C-R4: Use LSA, Word2Vec and LSTM-Sen2Vec to retrieve similar posts and get corresponding comments, LSA and Word2Vec to retrieve appropriate comments from all comments, combine and rank them with $Score^2_{q,c}(q, c)$ and get top-10 comments as results.

3. Nders-C-R3: Use graph-based algorithm TextRank with words as vertices in the graph, and use score $Rank_{TextRank}$ to rank the comment candidates from R4 and get top-10 comments.

4. Nders-C-R2: Use $Rank_{PI}$ as a ranking score to rank comment candidates from R4 and get top-10 comments.

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 candidates from R4 and get top-10 comments.

The official results of our five runs are shown in Table 5. Which shows that, with the use of Word2Vec and LSA model, R4 achieves best result in our five runs for Mean nG@1, that 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 by 2.02% for mean P+ and 1.26% for mean nERR@10 and both ranks 5th among 22 teams, with 0.77% slightly decreased for mean nG@1. It 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.

Moreover, we conduct a per-topic analysis. We classify 100 test posts according to their topics, including animal, philosophy, weather, entertainment, emotion, travel, technology, sports, art, diet and others. Then, for each topic, we calculate the mean value of each run for three evaluation metrics respectively, shown as Table5-Table7.

Results show that, the standard deviation of R5 (with LDA model) is bigger than R4 (with LSA model) and other runs. Though in some topics such as animal, sports, and diet, the mean value for each metric is significantly high than other topic, however, in other topics, the mean value

is rather small. It is because that the LDA vector is sparse, that is, most of entries of the vector is zero. Which results that the cosine similarity of two LDA vector is either close to 1 (high topic related) or close to 0 (topic not related).

It suggests that LDA model is unstable over topics. Which is also why we choose the candidates from R4 for further ranking by R1, R2 and R3.

4. CONCLUSIONS

In this paper, we propose an approach for STC-2 task of NTCIR-13. The LSA, Word2Vec and LSTM-Sen2Vec model are used to find similar posts. The LSA 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 LSA 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.

5. REFERENCES

[1] Lifeng Shang, Tetsuya Sakai, Zhengdong Lu, Hang Li, Ryuichiro Higashinaka, and Yusuke Miyao. Overview of the NTCIR-12 Short Text Conversation Task, Proceedings of NTCIR-12, 2016.

[2] Lifeng Shang, Tetsuya Sakai, Hang Li, Ryuichiro Higashinaka, Yusuke Miyao, Yuki Arase, and Masako Nomoto. Overview of the NTCIR-13 Short Text Conversation Task, Proceedings of NTCIR-13, 2017.

[3] Susan T. Dumais (2005). Latent Semantic Analysis. Annual Review of Information Science and Technology. 38: 188–230.

[4] Blei, David M, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. Journal of Machine Learning Research 3(2003):993-1022.

[5] Mikolov, Tomas, et al. Efficient Estimation of Word Representations in Vector Space. Computer Science (2013).

[6] Mikolov, Toma's. Statistical Language Models Based on Neural Networks. Ph.D. thesis, Brno University of Technology.(2012)

[7] Zaremba, Wojciech, I. Sutskever, and O. Vinyals. Recurrent Neural Network Regularization. Eprint Arxiv (2014).

[8] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[9] Sundermeyer, Martin, R. Schlüter, and H. Ney. LSTM Neural Networks for Language Modeling. Interspeech 2012:601-608.

[10] Graves, Alex. Generating Sequences With Recurrent Neural Networks. Computer Science (2013).

[11] Mihalcea, Rada, and P. Tarau. TextRank: Bringing Order into Texts. Unt Scholarly Works (2004):404-411.

[12] Brin, Sergey, and L. Page. The anatomy of a large-scale hypertextual Web search engine. International Conference on World Wide Web Elsevier Science Publishers B. V. 1998:107-117.

Table 5: Per-topic analysis for mean nG@1

Topic	Nders-C-R1	Nders-C-R2	Nders-C-R3	Nders-C-R4	Nders-C-R5
animal(4)	0.5000	0.4167	0.5417	0.5417	**0.6667**
philosophy(7)	0.3333	**0.4286**	0.3333	0.3333	0.1667
weather(3)	0.2778	0.2778	0.2778	0.2778	0.2778
entertainment(2)	**0.4167**	**0.4167**	0.3333	0.3333	0.1667
emotion(23)	0.4768	**0.5493**	0.5130	0.5275	0.5130
travel(4)	**0.5834**	0.5000	0.5000	0.5000	0.4583
technology(9)	**0.3889**	0.2963	0.3704	0.3704	0.2482
sports(5)	0.2867	0.4533	0.3600	0.3933	**0.6933**
art(5)	0.5000	0.5000	0.4000	**0.5667**	0.4333
diet(16)	0.5000	0.5000	0.5521	0.5521	**0.6771**
other(22)	**0.5091**	0.5015	0.4864	0.4864	0.3712
std.	0.1007	**0.0865**	0.0965	0.1040	0.1980

Table 6: Per-topic analysis for mean nERR@10

Topic	Nders-C-R1	Nders-C-R2	Nders-C-R3	Nders-C-R4	Nders-C-R5
animal(4)	0.6143	0.5868	0.6344	0.6359	**0.7459**
philosophy(7)	0.5015	**0.5489**	0.4704	0.4735	0.3721
weather(3)	0.4089	0.4034	0.4000	0.4000	**0.4893**
entertainment(2)	0.4932	**0.4988**	0.4551	0.4509	0.3394
emotion(23)	0.5854	0.6297	0.6050	0.6076	**0.6432**
travel(4)	**0.6811**	0.6487	0.6385	0.6435	0.5768
technology(9)	**0.5052**	0.4518	0.4532	0.4545	0.4465
sports(5)	0.4854	0.5466	0.5091	0.5203	**0.7779**
art(5)	0.6475	**0.6864**	0.5828	0.6726	0.6399
diet(16)	0.6537	0.6504	0.6976	0.6862	**0.7716**
other(22)	**0.5912**	0.5777	0.5715	0.5703	0.4722
std.	**0.0867**	0.0881	0.0949	0.1006	0.1577

Table 7: Per-topic analysis for mean P+

Topic	Nders-C-R1	Nders-C-R2	Nders-C-R3	Nders-C-R4	Nders-C-R5
animal(4)	0.5594	0.5082	0.5507	0.5655	**0.6805**
philosophy(7)	0.4711	**0.5295**	0.4256	0.4370	0.3556
weather(3)	0.3927	0.3910	0.3869	0.3869	**0.4749**
entertainment(2)	**0.4087**	0.3745	0.3517	0.3478	0.2511
emotion(23)	0.5500	0.5959	0.5526	0.5561	**0.6101**
travel(4)	**0.6352**	0.5919	0.5870	0.5870	0.5149
technology(9)	**0.5078**	0.4475	0.4106	0.4081	0.4057
sports(5)	0.4275	0.5312	0.4523	0.4647	**0.7344**
art(5)	0.5887	**0.6624**	0.5566	0.6512	0.5929
diet(16)	0.6080	0.6057	0.6580	0.6408	**0.6923**
other(22)	**0.5383**	0.5251	0.5362	0.5253	0.4705
std.	**0.0825**	0.0904	0.0964	0.1036	0.1521