RUCIR at NTCIR-13 WWW Task

Ming Yue[1], Zhicheng Dou
Beijing Key Laboratory of Big Data Management and Analysis Methods, China
Key Laboratory of Data Engineering and Knowledge Engineering, MOE, China
School of Information, Renmin University of China
[1]{yomin,dou}@ruc.edu.cn

ABSTRACT

In this paper, we present our approach in the We Want Web(WWW)[1] task of NTCIR-13, for both English and Chinese languages. We implement a ranking model for traditional re-ranking problems based on learning to rank. We first process the raw data and extract text features, match features, embedding features and semantic features for each query-document pair. Then we use LamdaMART[2] to train the ranking model and rank the documents by the ranking scores. Finally, we could get the ranking list.

Team Name

RUCIR

Subtasks

We Want Web (Chinese, English)

1. INTRODUCTION

In modern search engines, users type queries to a search box and search engines return relevant results. Whether the document list matches user's queries becomes an important issue. Thus document ranking is the core challenge for many applications of information retrieval. In the ranking task, given a set of web pages, we use a score function to get a ranking list. The relative order of documents may reflect their degrees of relevance to a user query.

There are two major approaches, which are traditional unsupervised approaches and learning to rank methods, to deal with the ranking problem. The traditional methods usually define the score function heuristically, such as TF-IDF and BM25[3] model. We only need to determine the function parameters and then calculate the document ranking score to create the final ranking list. The drawback of these models is that if a ranking function has only a small number of parameters, performance tuning can be done manually. However, if there are a large number of parameters, it will become very difficult and time-consuming.

Learning to rank approaches make good performance when applied to information retrieval. Assuming that each query is associated with a set of documents with relevance judgments. We could learn the parameters of a ranking function using the training data, such that the model can precisely score a document. When it comes to testing data, given a new query, the ranking function is used to create a ranked list based on scoring on the documents associated with that query.

These learning to rank algorithms can successfully utilize all kinds of features and automatically learn the optimal parameters. Many approaches have been proposed such as RankNet[4], ListNet[5] and so on. In recent years, neural network methods are also applied to information retrieval task. Such models e.g., ARC[6] and DSSM[7] focus on projecting the query and document to semantic vector space, and a score function calculates the similarity score between two vector embeddings.

In the We Want Web subtask[1], for both Chinese and English task, the official data consists of 100 queries and top 1,000 documents for each query, which are obtained by baseline retrieval systems, based on Solr[12]. Our goal is to use some training data to re-rank the baseline runs to get better search results.

We choose to use learning to rank approaches to do the We Want Web(WWW) task. In section 2, we illustrate algorithm details on WWW task from data preparation to model training and evaluation metrics. In section 3, we show evaluation results of our submitted runs and analyze the results. In Section 4, we come to conclude about this WWW task.

2. WE WANT WEB TASK

In this section, we introduce our work on WWW task. Data flow are described as follows. First, we need to prepare the training data from previous TREC task and NTCIR task. Second, we extract four types of features for each labeled query-document pair. Third, we use a popular learning to rank algorithm, named LambdaMART, to train and validate the ranking model. Finally, we process the test data and create the final ranking list. Figure 1 shows the whole process.

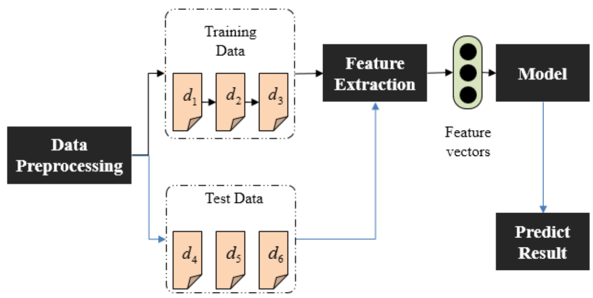

Figure 1: Dataflow Overview

2.1 Data Set

2.1.1 *Official Dataset*

We use the official datasets for this ranking task. For Chinese Subtask, we use the SogouT-16[1] as the document collection. SogouT-16[10] contains about 1.17B Web pages, but we only index part of the "Category B" version of SogouT-16. Although user behaviour data is available, due to the tight schedule of the task, we do not use them in this ranking task. For the English Task, we adopt the ClueWeb12-B13 as the document collection.

2.1.2 *Training Dataset*

Since official datasets do not include abundant training data, we need to find some other data for supplement. For English task, we use the labeled data from previous TREC competitions. We collect about 200 queries and their relevance judgment files from TREC09 to TREC14 competitions. We parse and index Category A of ClueWeb09 document collection, which contains about 50,000,000 web pages, for basic search utilization and feature extraction. Similarly we do the same thing on the Category B of ClueWeb12 dataset. Further, we also use the anchor text information, spam collection and link graph of both ClueWeb09 and ClueWeb12 for more analysis. All the resources we use in English subtask are summarized as follows:

- Document Collection (ClueWeb09, ClueWeb12)
- Relevance Judgement (Trec09 - Trec14)
- Anchor Text (ClueWeb09, ClueWeb12)
- Link Graph (ClueWeb09, ClueWeb12)
- Spam Collection (ClueWeb09, ClueWeb12)

For Chinese task, we only use the labeled data from previous NTCIR IMINE tasks and the official relevant judgments. We randomly sample about 20,000,000 web pages from Category B of SogouT-16 and merge a little part of SogouT-08 web pages together, indexing them by Solr for basic use. All the resources are summarized as follows:

- Document Collection (SogouT-08, SogouT-16)
- Relevance Judgments (NTCIR09-11, Official)

2.2 Feature Extraction

Ranking task is somewhat a machine learning problem. We need to extract various kinds of features to represent a query-document pair for model training. Each training sample is represented by a multi-dimensional feature vector, and each dimension of the vector is a feature, which indicates how important the document is with respect to the query. We hope to cover as many features proposed in IR papers and conferences as possible. In this paper, we use four kinds of features, which are traditional text relevant features, match features, embedding features and semantic neural network features.

The format of the final feature file is the same as SVM-Rank format. Each of the following lines represents one training example and the line format is: relevance label and query id, followed by feature id and its value, as shown in Figure 2.

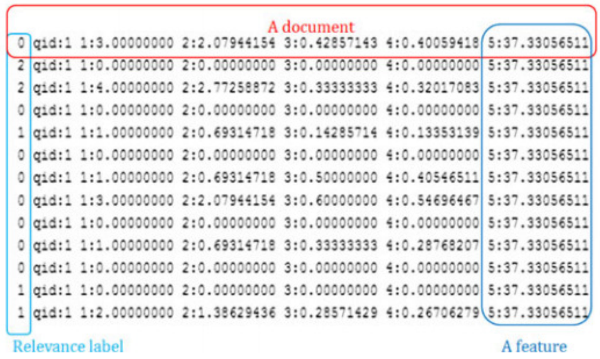

Figure 2: Feature File Format

Table 1: Text Relevant Features

Names	Description	#Features
Boolean	Boolean model	5
TF-IDF	TF-IDF model	5
BM25	BM25 default parameters	5
LMIR	Dirichlet smoothing	5
IDF	Sum of IDF	5
TF	Sum of term frequency	5
DL	Document length	5
Spam	Spam score	1
Pagerank	PageRank score	1
# Inlinks	Number of inlinks	1
# Outlinks	Number of outlinks	1

2.2.1 *Text Relevance Features*

Text Relevance Features are the most traditional features that can be used in the learning task. We classify the features into two classes. One class is query-document features which are dependent on both the query and the document, such as BM25 feature. The other class is document features, which means that the feature only depends on the document, such as pagerank[11] and document length.

In addition, we consider five fields of a document: body, anchor, title, URL and the whole document. Because we think many score functions such as BM25 may have different effects on different fields, which could affect the final ranking result. Table 1 shows the most common traditional text relevant features. The description column in Table 1 gives a brief introduction on these features.

2.2.2 *Match Features*

Match features are corresponding to occurrence of query items in the document. There are two match feature classes: perfect match and complete match.

Perfect Match. As mentioned above, when a query is split into several terms, we will find whether these terms occur in the document. And at the same time, we focus on whether these terms occur continuously as a integrated phrase in the document. Supposing that we have a query "I like you", *perfect match* means whether we could find the substring "I like you" in the document. This could be a strong signal indicating that the document is relevant to the query.

Complete Match. Different from *perfect match*, *complete match* has weak conditions on whether query terms must occur as an integrated phrase. Taking query "I like you" as an example, *complete match* requires that terms "I", "like" and "you" should occur in the document, but there could be span words between them. The document "I really like somebody and you." holds this scenario. *Complete match* somewhat likes the span-based feature, so that we calculate the maximum span between query terms. We use 10 buckets to convert continuous span value to category value according to span value statistics. Thus, we could get one-hot encoded complete features for further use.

In the experiment, we find the *perfect match* and *complete match* in the title and body field. For English task, we also calculate the *perfect match* feature in the URL and anchor text.

2.2.3 Embedding Features

Although text relevant features and match features are useful in ranking task, but they lack the semantic meanings of query and document. So we introduce the embedding features and semantic neural network features next section to represent the query and document in a semantic vector space. Embedding features in this section focus on word vector representation, such as Word2vec[8] and Doc2vec[9]. The following two parts introduce how we use these embedding models to extract features in detail.

Word2vec. Word2vec was created by a team of researchers led by Tomas Mikolov at Google. Word2vec uses two-layer neural networks to train a large corpus to get high dimensional word vectors, typically of several hundred dimensions, by using rich context information.

Here, we first get each word vector from corpus, then we create the distributed representations of queries and documents, and finally we calculate the similarity score between each query and document pair as one embedding feature.

Initially we use the Word2vec tool from Google to train word vectors both on title and body using our huge document collection, as mentioned in Section 2.1.

Next, we use pre-trained word vectors to obtain the distributed representation of the query or document. Because documents are composed of terms, and each term can be represented by its vector, we use mean, max, min function on terms to get document embedding vectors. We use $V_d \in \mathbf{R}^m$ to note the final representation, which the $i-th$ dimension would be calculated as follows:

$$V_{di} = \frac{1}{n}\sum_{j=1}^{n} Term_{ji}$$

$$V_{di} = max(Term_{ji}), j \in [1...n]$$

$$V_{di} = min(Term_{ji}), j \in [1...n]$$

where n means the number of terms in the document; $Term_{ji}$ denotes the ith position of the word vector $Term_j$. Because the number of terms in one document is too large, we extract several snippets to replace the whole document.

Finally, we use cosine distance as the similarity score between each query and document pair. After we get representations of the query and document, noted as V_q and V_d, we can calculated feature value as follows:

$$score(q, d) = cosine(V_q, V_d)$$

Doc2vec. An extension model of the Word2vec to construct embeddings from entire documents, rather than the individual words, has been proposed. This extension is called Paragraph2vec or Doc2vec.

As shown in Figure 3, every document is mapped to a fixed length vector, represented by a column in matrix D and every word is also mapped to a unique vector, represented by a column in matrix W. The document vector and word vectors are averaged or concatenated to predict the next word in a context. The contexts are fixed-length sliding window sentences and sample from the document. The document vector is shared across all contexts generated from the same document but not across documents. We can consider a document vector as a specific word vector that "stores" the missing information from current context.

The document vectors and word vectors are trained using stochastic gradient descent and the gradient is obtained via back propagation. Then the following calculating feature value steps are exactly the same as what Word2vec method does mentioned before.

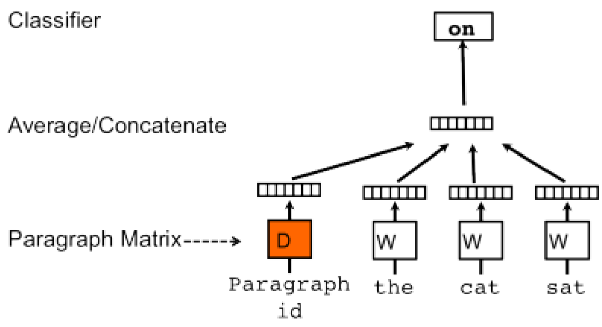

Figure 3: Doc2vec Framework

2.2.4 Semantic Neural Network Features

In previous section, we use Word2vec or Doc2vec to obtain the distributed representation of a document. Currently, we use deep learning models, such as CNN and RNN, to get deep semantic relationships between word sequences.

This paper[6] proposed a new convolutional architecture for modeling sentences. It takes the embedding vectors of words in the sentence as input, and summarizes the meaning of a sentence through layers of convolution and pooling. Finally we could get a fixed length vector representation in the final layer. Based on this new architecture, they propose a related convolutional architecture, namely Architecture-I(ARC-I), for matching two sentences. ARC-I, as illustrated in Figure 4, takes a conventional approach: It first generates the representation of each sentence, and then compares the representations for the two sentences with a multi-layer perceptron. We use this model to calculate the semantic similarity score as our features.

2.3 Model Training

There are a lot of learning to rank algorithms that can perfectly solve this task. Due to limited time, we simply just take LambdaMART, a very stable approach which can directly optimize evaluation metrics, to train, validate and test on the dataset. We use a third-party package, Ranklib, to implement our offline re-ranking system.

We partition the training data into five equal parts for 5-fold cross validation. We select 4 folds for training and the

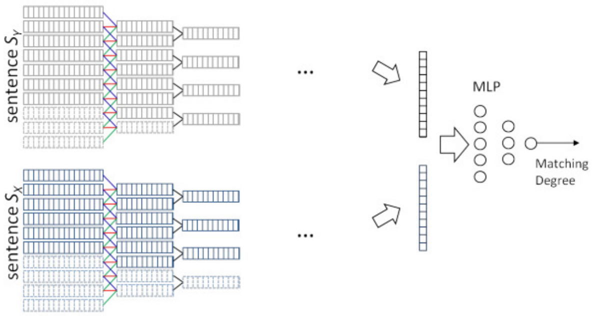

Figure 4: ARC-I Architecture

other as validation data each time. The training set is used to learn ranking models. The validation set is used to tune the hyper parameters of the learning algorithms, such as the number of iterations in LambdaMART. After obtaining the hyper parameters, we finally train a model on these five folds for further testing.

Feature selection has been proved useful for improving the model performance. However, feature selection is a time-consuming engineering. Thus, we do not leverage feature selection in this task.

2.4 Evaluation Metrics

In both Chinese and English tasks, we use nDCG@K for evaluation, which is the most common metric used in the ranking task. nDCG@k is a measure for evaluating top k positions of a ranked list using multiple levels of relevance judgment. It is defined as follows:

$$nDCG@k = N_k^{-1} \sum_{j=1}^{n} g(r_j)d(j)$$

where k means the top k results of ranking list, N_k denotes the maximum of $\sum_{j=1}^{n} g(r_j)d(j)$, r_j denotes the relevance level of the document ranked at the j-th position; $g(r_j)$ denotes a gain and $d(j)$ denotes a discount function. In the experiment, we adopt nDCG@1, nDCG@5, nDCG@10 to evaluate the ranking result. For selecting the best parameter, we use the mean average of the above three metrics.

2.5 Experiments

2.5.1 submitted runs

We submit the following five runs for both Chinese and English We Want Web task:

- RUCIR-C/E-NU-Base-1: text relevant features.
- RUCIR-C/E-NU-Base-2: text relevant features, match features.
- RUCIR-C/E-NU-Base-3: text relevant features, match features, embedding features.
- RUCIR-C/E-NU-Base-4: text relevant features, match features, semantic neural network features.
- RUCIR-C/E-NU-Base-5: text relevant features, match features, embedding features, semantic neural network features.

2.5.2 Experimental Results

Table 2 and Table 3 show the evaluation results of our submitted runs. We see that the traditional text relevance features achieve the best performance in both Chinese and English tasks. In Chinese task, we are one of the top performance teams.

RUCIR--NU-Base-1 vs. RUCIR-*-NU-Base-2.* The result shows that text relevant features are proved to be efficient for ranking tasks. Perfect match features shows strong relevant signal when we debug on the ranking algorithm. While complete match features' performance seems to be not stable when validating on different datasets. Maybe we should look more into the span distance partition algorithm in complete match method.

RUCIR--NU-Base-2 vs. RUCIR-*-NU-Base-3.* We find that it may not perform well when using word embedding features in both Chinese and English tasks. We randomly select several similar document pairs to test the pre-trained embeddings. We find that both the doc2vec and word2vec vectors similarity scores are not as high as expectation. The possible reason is that there is much noise when we parse the raw web pages to plain text.

RUCIR--NU-Base-3 vs. RUCIR-*-NU-Base-4.* We can see that semantic neural network features really do some work on improving the results. ARC-I network structure could be considered as another distributed representation of documents. Although we find that the word vectors we trained are not as good as we expect, ARC-I captures more information on interaction between document pairs such as click behaviour. And thus it could perform better than pure embedding methods.

Table 2: Chinese We Want Web results. Mean score of each Metric.

Run name	nDCG@10	Q@10	nERR@10
RUCIR-C-NU-Base-1	0.6323	0.6449	0.7771
RUCIR-C-NU-Base-2	0.6241	0.6448	0.7597
RUCIR-C-NU-Base-3	0.5361	0.5407	0.6767
RUCIR-C-NU-Base-4	0.5873	0.6049	0.7217
RUCIR-C-NU-Base-5	0.5827	0.5890	0.7132

Table 3: English We Want Web results. Mean score of each Metric

Run name	nDCG@10	Q@10	nERR@10
RUCIR-E-NU-Base-1	0.5254	0.5135	0.6988
RUCIR-E-NU-Base-2	0.4207	0.4050	0.5795
RUCIR-E-NU-Base-3	0.4516	0.4402	0.5917
RUCIR-E-NU-Base-4	0.3843	0.3859	0.5343
RUCIR-E-NU-Base-5	0.3885	0.3813	0.5292

3. CONCLUSIONS

In this paper, we describe our approaches for the WWW task in NTCIR-13. In Chinese subtask, we achieve great performances on nDCG@10, Q@10 and nERR@10. However, some enhanced methods performed not as well as our expect. The reason is that embedding and neural network approaches need large training data while we do not have enough data. In the future we will do more work to handle this problem.

4. REFERENCES

[1] C Luo, T Sakai, Y Liu, Z Dou, C Xiong, J Xu. Overview of the NTCIR-13 We Want Web Task. In *Proceedings of NTCIR-13 workshop* , 2017

[2] Burges, Chris J. C. From RankNet to LambdaRank to LambdaMART: An Overview. *Microsoft Research*, June 2010.

[3] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In *Proc. the 3rd TREC 1994*, pages 109–126, 1995.

[4] C.J.C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton and G. Hullender Learning to Rank using Gradient Descent. In *Proceedings of the Twenty Second International Conference on Machine Learning*, 2005.

[5] Z. Cao, T. Qin, T. Liu, M. Tsai, H. Li. Learning to Rank: From Pairwise Approach to Listwise Approach In *ICML*, 2009.

[6] Hu B, Lu Z, Li H, Chen Q. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*(pp. 2042–2050).

[7] Huang PS, He X, Gao J, Deng L, Acero A, Heck L. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, 2013

[8] Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 2013

[9] Le Q, Mikolov T. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning* , 2014

[10] C Luo, Y Zheng, Y Liu, X Wang, J Xu, M Zhang, S Ma. SogouT-16: A New Web Corpus to Embrace IR Research. In *In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*, ACM, New York, NY, USA, 1233-1236.

[11] Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry The PageRank Citation Ranking: Bringing Order to the Web. In *Stanford InfoLab*, 1999

[12] http://lucene.apache.org/solr/