Generalized Learning of Neural Network based Semantic Similarity Models and its Application in Movie Search

Xugang Ye, Zijie Qi, Xinying Song, Xiaodong He, Dan Massey
Microsoft
Bellevue, USA
{xugangye, zijieqi, xinson, xiaohe, danmass}@microsoft.com

***Abstract*—Modeling text semantic similarity via neural network approaches has significantly improved performance on a set of information retrieval tasks in recent studies. However these neural network based latent semantic models are mostly trained by using simple user behavior logging data such as clicked (query, document)-pairs, and all the clicked pairs are assumed to be uniformly positive examples. Therefore, the existing method for learning the model parameters does not differentiate data samples that might reflect different relevance information. In this paper, we relax this assumption and propose a new learning method through a generalized loss function to capture the subtle relevance differences of training samples when a more granular label structure is available. We have applied it to the Xbox One's movie search task where session-based user behavior information is available and the granular relevance differences of training samples are derived from the session logs. Compared with the existing method, our new generalized loss function has demonstrated superior test performance measured by several user-engagement metrics. It also yields significant performance lift when the score computed from our model is used as a semantic similarity feature in the gradient boosted decision tree model which is widely used in modern search engines.**

Keywords—neural network; semantic model; loss function; click logs; movie search

I. Introduction

Nowadays search engines are heavily relied on for retrieving relevant information for users, and search engines that can understand the search intent behind the words of a query despite language divergence are highly in demand. However, this presents a great challenge. Unlike term or lexical matching, which is straightforward and easy to implement, building a search engine that understands the intent and contextual meaning of the query is difficult, especially when the query is short and ambiguous. In order to address this problem, many latent semantic models have been proposed during the past decade. Let's review some of the major techniques presented in the literature.

A. Latent Semantic Models

Latent Semantic Analysis (LSA) [7][8] is a straightforward and well-known latent semantic model. It reconstructs the term-document matrix by using low rank matrix approximation such that both the terms and the documents can be mapped to a low dimensional space. However, the mapping is done by a linear projection. Nonlinear methods include popular topic models such as the probabilistic latent semantic indexing (PLSI) [12] and the Latent Dirichlet allocation (LDA) [2], with each being a generative model and having a strong probability foundation. PLSI assumes that the document index, which has a multinomial prior, generates a latent topic and the topic in turn generates a word. LDA assumes that a word is generated by a latent topic, and the topic is a sample of a multinomial distribution that has a Dirichlet prior. By using either PLSI or LDA, a document's representation at the topic level can be computed [11]. One important application of the latent semantic models is to fulfil the needs of semantic matching for search engines by calculating the similarity between the documents at the topic or semantic level. Recently, some semantic models were proposed for search specifically. For examples, the coupled probabilistic latent analysis (CPLSA) by Platt, Toutanova and Yih [17] is an extension of the PLSI, the Bi-Lingual Topic Model (BLTM) by Gao, Toutanova and Yih [9] is an extension of the LDA, and both of them can calculate the query-document similarity at the topic level.

B. Neural Network Models

Another set of latent semantic models are neural network based. It has been shown that a neural network with multiple hidden layers can discover even more sophisticated latent structures than a neural network with a single hidden layer [1][19]. Therefore, recently a series of latent semantic models with deep neural network structure have been proposed to model complex concepts and hidden hierarchical structures [1][10][13][16][19][20]. The Semantic Hashing method by Salakhutdinov and Hinton [19] was designed to project a bag-of-words based term vector to a binary code vector by an auto encoder that minimizes the reconstruction error. Recently, a deep Structured Semantic Model (DSSM) was developed by Huang, He, Gao and Deng [13] to model the semantic similarity between a query and a document for the task of web search. More recently, Shen, He, Gao, Deng and Mesnil [20] extended the DSSM to the convolutional latent semantic model (CLSM) to capture important contextual information without making a strong bag-of-words assumption.

Compared with the previous latent semantic models, the key distinct feature of DSSM and CLSM is that they are task-specific supervised learning algorithms. Both DSSM and CLSM were originally designed for web search and they were trained by using the clicked (query, document)-pairs. On the contrary, the previous latent semantic models are based on unsupervised learning and the semantic similarity computed from any of these models is not learned from the labels. It has been reported in [13][20] that when used as a single-feature ranker for web search, both DSSM and CLSM significantly outperform other latent semantic models such as LSA, PLSI, BLTM in terms of the NDCG (Normalized Discounted Cumulative Gain [14]) measurements using human labels. Although CLSM has higher NDCG values than DSSM, due to the convolutional neural network structure, the CLSM's computational cost in scoring is much higher than DSSM, which could be a concern for the online search system. Besides the web search task, DSSM and CLSM can also be applied to a broader set of applications such as word embedding [21] and question-answering [22], etc.

C. *Loss Function*

Despite the superior performance of DSSM and CLSM, these models treat all the clicked (query, document)-pairs uniformly as positive examples. Therefore, the current method for learning the model parameters does not differentiate data samples that might reflect different relevance information. In other words, there is no differentiation between two clicked documents under the same query, with one being more relevant and the other being less relevant. In this paper, we propose a generalized loss function that can incorporate the subtle relevance differences among the documents for learning the model parameters. Our experimental results have shown that the new method can significantly improve the ranking results on the movie search tasks.

Our method requires fine-grained relevance labels. Some commercial search engines like Bing has already utilized multi-level relevance information. There are usually two kinds of resources: human judgments and search logs. The human labels for web search usually have 5 coarse grained relevance levels: Perfect, Excellent, Good, Fair, and Bad. When this type of labels are taken, each category will be converted into an appropriate number (the more relevant the label represents, the greater the number is). The labels constructed from the search logs often take various forms of click-likelihood, which have numerical values. Although the commercial search engines have already used fine-grained relevance labels, none of neural network based latent sematic models has done so. This is the first study that investigates using fine-grained relevance labels to improve the neural-network-based latent semantic models.

D. *Model Score as Ranking Feature*

Since the semantic similarity score computed from a neural-network-based latent semantic model can be viewed as a feature, it is worthwhile considering how a commercial search engine can benefit from this feature. Currently the LambdaMart algorithm [4], which is an extension of the LambdaRank algorithm [4], is widely used as a core ranking algorithm of many commercial search engines including Bing. The LambdaMart is a gradient boosted decision tree model that takes as many features as it can and selects the important ones. Usually, many sophistic features are manually built for the LambdaMart based on term or lexical matching. An interesting question is how much improvement there could be if the semantic similarity score is added as a new feature to the LambdaMart. Our experimental results have shown that the semantic similarity score computed from our model not only outperforms the semantic similarity scores computed from previous state-of-the-art models such as DSSM, but also further improves the overall performance of a strong LambdaMart-based ranker when used as an additional feature.

E. *Movie Search*

It's very expensive to obtain high quality human labels on a large scale. As a result, both DSSM and CLSM for the web search task were trained from the click-through data and evaluated using the human labels. Moreover, if there are only a very limited number of judges, the bias is a serious concern. Although click signals can easily scale up, they are very noisy. However, we found that for media search, noise in click information is easier to handle, and labels can be built from the click-through data with good quality comparable to human labels. Therefore, as the first shot, we selected the media search domain and used movie search logs to experiment on our idea.

We extracted the movie search logs from the Xbox One, a very popular entertainment platform. The data has an advantage that each logged query session contains a user ID. Therefore, we can calculate how many distinct users have clicked a movie under a query in a period of time and use it as the label for a (query, movie)-pair. This is an aggregated number that is robust to noise, easy to scale up and calculate. Our experiments have shown that the labels generated in this way are highly consistent with the human labels and they can be easily built into our generalized loss function. Obviously, this advantage also widely exists in many other online video service platforms such as YouTube, Netflix, Amazon Video and Hulu etc. Therefore, the method for generating labels from the Xbox One's search logs can also be used for generating the same kind of labels for those platforms. We should point out that our generalized loss function is not limited to the specific domain of media search. It can be used broadly as long as fine-grained labels can be built.

F. *Organization of the Paper*

The rest of the paper is organized as follows. We first describe our generalized loss function for the neural-network-based latent semantic models and provide an analysis of its probability foundation. We then define our model, with the same architecture as the DSSM model in [13], but it is learned by minimizing the proposed generalized loss function. We also describe the corresponding new gradient computation method and the dimension reduction technique. The reason why we choose DSSM's architecture is because it has much lower computational cost in scoring than CLSM and hence it's easier to implement for a commercial search engine. Replacing CLSM's loss function will be studied in future work. After defining two types of evaluation metrics in our experiments, we present the results of applying our model to the task of movie search and compare it against existing models on various benchmarks. In evaluation, we not only introduce the effectiveness of our model in the single-feature-ranking setting, but also present the results of adding the semantic similarity score computed from our model to the LambdaMart as a new feature. Our model leads to significant improvement in both settings, which demonstrates the effectiveness of the proposed method. In the end, we draw the conclusion and suggest future research directions.

II. OUR MODEL

A. *Generalized Loss Function*

The main contribution of this work is the generalization of the loss function for learning the neural network based semantic similarity models. Extended from the loss function originally proposed in [13], the generalized loss function takes into account fine-grained relevance labels and captures the subtle relevance difference of different data samples. Suppose $\{(q_i, d_i): i = 1,2,\dots,n\}$ are n (q,d)-pairs such that d_i is clicked under q_i. To learn a semantic similarity model, the DSSM in [13] aims to minimize

$$L_1(\Lambda) = -\sum_{i=1}^{n}[\ln P(d_i|q_i;\Lambda)], \quad (1)$$

where $P(d_i|q_i;\Lambda)$ is the parameterized conditional probability that document d_i is relevant given query q_i and Λ denotes the set of model parameters. Minimizing this loss function is interpreted as maximizing the joint probability that (q_1, d_1), …, (q_n, d_n) are relevant pairs, with the assumption that they are independent of each other. Note that in the objective function in (1) "clicked" is treated as "relevant" regardless how many different people clicked d_i under q_i. In order to take into account the various relevance levels reflected by different click signals for different clicked pairs, we proposed a generalized loss function. Suppose for each i, the clicked pair (q_i, d_i) is labeled y_i, where $0 \leq y_i \leq 1$ and y_i is a probabilistic measure of the relevance of d_i to q_i. Our generalized loss function is expressed as

$$L_2(\Lambda) = -\sum_{i=1}^{n}\left[y_i \ln P(d_i|q_i;\Lambda) + (1-y_i)\ln\left(1 - P(d_i|q_i;\Lambda)\right)\right] \quad (2)$$

Clearly, when $y_i = 1$ for all i, (2) reduces to (1). To interpret this loss function, imagine that there are m users. For the i-th pair (q_i, d_i), the relevance probability is $P(d_i|q_i;\Lambda)$. Suppose relevance leads to click(s), and y_i is the portion of the m users who clicked d_i given q_i, then the probability that there are my_i users who click d_i under q_i is

$$\pi_i(\Lambda) = \binom{m}{my_i} P(d_i|q_i;\Lambda)^{my_i}\left(1 - P(d_i|q_i;\Lambda)\right)^{m-my_i}. \quad (3)$$

Assuming the clicks are independent of each other, the joint probability that there are my_i users who click d_i under q_i for $i = 1,\dots,n$ is

$$\prod_{i=1}^{n}\pi_i(\Lambda) \propto \prod_{i=1}^{n} P(d_i|q_i;\Lambda)^{my_i}\left(1 - P(d_i|q_i;\Lambda)\right)^{m-my_i}. \quad (4)$$

By taking the negative natural logarithm of (4), we have

$$-\sum_{i=1}^{n}\ln \pi_i(\Lambda) = -mL_2(\Lambda) + \text{Const.} \quad (5)$$

Therefore, minimizing L_2 in (2) is equivalent to maximizing the joint probability in (4). And this joint probability takes into account the probabilistic labels $y_1, \dots, y_n$.

B. *Analysis*

To illustrate the benefit of generalizing L_1 in (1) into L_2 in (2), let's consider the test accuracy of the prediction.

Let $\vec{P}$ be a probabilistic prediction vector for a collection of test cases. Let $\vec{y}'$ be an approximated target vector, and recall $\vec{y}$ is the true target vector. Note that $0 \leq y_i' \leq 1$ for all i. Assume $y_i' = A(y_i)$ for all i, where $A: [0,1] \to [0,1]$ is an approximation function that satisfies

(i) A is monotonically increasing;

(ii) $A(0) = 0, A(1) = 1, \lim_{y \to 0} A(y)\ln y = 0$.

The binary labels can be viewed as a special A such that $A(y) = 1$ if $y > y^*$; $A(y) = 0$ otherwise, where y^* is the cut-off and $0 \leq y^* \leq 1$. More generally, the layered labels (e.g., "Perfect", "Excellent", "Good", "Fair", "Bad") can be viewed as having multiple cut-offs. By using the concept of Kullback–Leibler divergence [15] (or KL-distance), we can show how much accuracy could be lost when $\vec{y}' = A(\vec{y})$ is used to approximate $\vec{y}$.

We first consider the loss of having prediction $\vec{P}$ when the true target $\vec{y}$ is given:

$$L(\vec{P};\vec{y}) = -\sum_i[y_i \ln P_i + (1-y_i)\ln(1-P_i)]. \quad (6)$$

Note that this loss is random since $\vec{P}$ is random. We further consider the expectation of this loss, denoted as $E\left[L(\vec{P};\vec{y})\right]$. Note that

$$\begin{aligned} & E\left[L(\vec{P};\vec{y})\right] \\ &= -\sum_i[y_i E(\ln P_i) + (1-y_i)E(\ln(1-P_i))] \\ &\geq -\sum_i[y_i \ln E(P_i) + (1-y_i)\ln(1-E(P_i))] \\ &\geq -\sum_i[y_i \ln y_i + (1-y_i)\ln(1-y_i)], \end{aligned} \quad (7)$$

where the first "$\geq$" is by Jensen's inequality and the second "$\geq$" is due to the fact that $y_i \ln x + (1-y_i)\ln(1-x)$ is maximized over $0 \leq x \leq 1$ when $x = y_i$. The lower bound can be reached if and only if $\vec{P}$ equals the constant $\vec{y}$. This condition seems to be too strong since no prediction can be expected to have 100% accuracy. However, the necessary condition $E(\vec{P}) = \vec{y}$ for reaching the lower bound is realistic. We define a model as *consistent* if the expected value of its prediction equals its target.

We're now ready to show how much accuracy could be lost when a consistent model generates a prediction $\vec{P}$ of the approximated target $\vec{y}' = A(\vec{y})$. By consistency definition, we have $E(\vec{P}) = \vec{y}'$. We are interested in the quantity $E\left[KL(\vec{P}||\vec{y})\right]$, which is the expected KL-distance between the prediction $\vec{P}$ and the true target $\vec{y}$. We have

$$\begin{aligned} & E\left[KL(\vec{P}||\vec{y})\right] \\ &= E[-\sum_i P_i(\ln y_i - \ln P_i)] \\ &= E[-\sum_i P_i(\ln y_i - \ln y_i' + \ln y_i' - \ln P_i)] \\ &= E[-\sum_i P_i(\ln y_i - \ln y_i')] + E[-\sum_i P_i(\ln y_i' - \ln P_i)] \\ &= -\sum_i E(P_i)(\ln y_i - \ln y_i') + E[KL(\vec{P}||\vec{y}')] \\ &= -\sum_i y_i'(\ln y_i - \ln y_i') + E[KL(\vec{P}||\vec{y}')] \\ &= KL(\vec{y}'||\vec{y}) + E[KL(\vec{P}||\vec{y}')]. \end{aligned} \quad (8)$$

Hence, by the KL-distance measure, predicting $\vec{y}'$ yields the loss of accuracy that is at least $KL(\vec{y}'||\vec{y})$. That is to say, even if the model can learn its target with 100% accuracy such that the second term vanishes, there is still the first term remaining that is completely due to the label error. If we can improve the labels, then we can improve the prediction independent of the learning model. Back to the benefit of generalizing L_1 in (1) into L_2 in (2), since L_1 corresponds to the extreme case that all the clicked pairs are labeled 1, it can be expected that better labels could be built to obtain better ranking results.

III. Learning Model

The learning model is essentially the parameterization structure of the relevance probability $P(d|q)$ for the query q and document d. As mentioned earlier, we adopted the structure of the DSSM in [13]. At first, $P(d|q)$ is defined as a normalized exponential of the semantic similarity function denoted as $R(q,d)$. Then $R(q,d)$ is parameterized via two neural networks, with one for q and the other for d. We can show that for parameter estimation, the formula for computing the gradient of L_1 in (1) only needs slight changes to fit for L_2 in (2).

A. Relevance Probability

The softmax form of the parameterized relevance probability $P(d|q;\Lambda)$ can be expressed as

$$P(d|q;\Lambda) = \frac{\exp(\gamma R(q,d;\Lambda))}{\exp(\gamma R(q,d;\Lambda)) + \sum_{d' \in D^-} \exp(\gamma R(q,d';\Lambda))}, \quad (9)$$

where $\gamma > 0$ is a pre-determined smooth parameter and D^- is the set of all irrelevant documents to be ranked under q. In practice, for many queries, there are very few or no irrelevant documents to be ranked, as a result, D^- is approximated by randomly choosing $J(\geq 4)$ unclicked documents under q_i.

B. Semantic Similarity

The parameterized semantic similarity function $R(q,d;\Lambda)$ is defined in the form of the cosine similarity:

$$R(q,d;\Lambda) = \frac{\vec{x}^{(q)T}\vec{x}^{(d)}}{||\vec{x}^{(q)}|| \cdot ||\vec{x}^{(d)}||}, \quad (10)$$

where $\vec{x}^{(q)} = u(q;\Lambda^{(q)})$ and $\vec{x}^{(d)} = v(d;\Lambda^{(d)})$ are the semantic vectors of q and d respectively, and $\Lambda^{(q)}$ and $\Lambda^{(d)}$ are parts of parameter set Λ corresponding to q and d respectively. The two functions u and v are represented by two neural networks. For both nets, $tanh$ function is used as the activation function. That is if we denote the k-th layer as $(l_1^k, l_2^k, \ldots, l_{n_k}^k)$ and the $(k+1)$-th layer as $(l_1^{k+1}, l_2^{k+1}, \ldots, l_{n_{k+1}}^{k+1})$, then for each $i = 1, \ldots, n_{k+1}$,

$$l_i^{k+1} = \frac{1-\exp(-2a_i^k)}{1+\exp(-2a_i^k)}, \quad (11)$$

where $a_i^k = \sum_{j=1}^{n_k} w_{j,i}^k l_j^k + w_{0,i}^k$. Note that the last layers for q and d are $\vec{x}^{(q)}$ and $\vec{x}^{(d)}$ respectively. The structure is illustrated in the following Fig. 1.

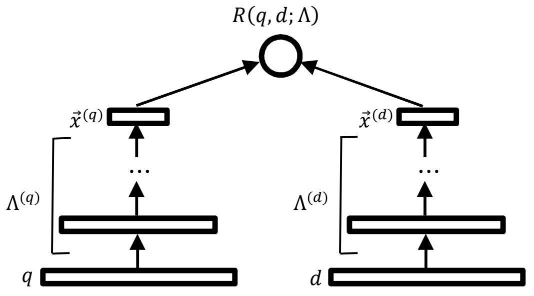


Fig. 1. Illustration of the neural network structure for computing $R(q,d;\Lambda)$. For both the query q and document d, it maps high dimensional sparse bag-of-words term vectors into low dimensional dense semantic vectors.

C. Parameter Estimation

We applied the gradient descent method to estimate the parameters. That is

$$\Lambda^{(t)} = \Lambda^{(t-1)} - \eta_t \nabla_\Lambda L(\Lambda), \quad (12)$$

where $\eta_t (> 0)$ is the learning rate and $\nabla_\Lambda L(\Lambda)$ is the gradient of $L(\Lambda)$ with respect to Λ. To calculate the gradient of the loss function in (2), we first express $P(d_i|q_i;\Lambda)$ as

$$P(d_i|q_i;\Lambda) = \frac{1}{1+\sum_{d \in D_i^-} \exp(-\gamma \Delta_d^i)}, \quad (13)$$

where $\Delta_d^i = R(q_i,d_i;\Lambda) - R(q_i,d;\Lambda)$. Then

$$\nabla_\Lambda \log P(d_i|q_i;\Lambda) = -\sum_{d \in D_i^-} \alpha_d^i \cdot \nabla_\Lambda \Delta_d^i, \quad (14)$$

where $\alpha_d^i = \frac{-\gamma \exp(-\gamma \Delta_d^i)}{1+\sum_{d' \in D_i^-} \exp(-\gamma \Delta_{d'}^i)}$, and

$$\nabla_\Lambda \log\left(1 - P(d_i|q_i;\Lambda)\right) = \sum_{d \in D_i^-} (\beta_d^i - \alpha_d^i) \cdot \nabla_\Lambda \Delta_d^i, \quad (15)$$

where $\beta_d^i = \frac{-\gamma \exp(-\gamma \Delta_d^i)}{\sum_{d' \in D_i^-} \exp(-\gamma \Delta_{d'}^i)}$. Therefore,

$$\nabla_\Lambda L(\Lambda) = \sum_{i=1}^{n} \sum_{d \in D_i^-} \left[\alpha_d^i - (1-y_i)\beta_d^i\right] \cdot \nabla_\Lambda \Delta_d^i. \quad (16)$$

In the special case that $y_i = 1$ for all i, this formula reduces to the same form as (1) used in [13][20].

D. Dimension Reduction

Since the dimension of the sparse bag-of-words term vector representation of an input text stream can be very high due to a vast vocabulary size and misspellings, we apply the letter-tri-gram (LTG) based text stream representation for the purpose of the dimension reduction [13]. To illustrate the idea, consider the English text stream "2014 Sci-Fi Movies". It's first converted to "#2014# #sci# #fi# #movies#", and then broken into "#20 201 014 14# #sc sci ci# #fi fi# #mo mov ovi vie ies es#", which is the final LTG-sequence. If we only include the 26 lower case English letters a-z and the 10 digits 0-9, then the size of the LTG-dictionary will be $36^3 + 2 \times 36^2 + 36 = 49{,}284$. In general, the size can be expressed as $l^3 + 2 \times l^2 + l$, where l is number of valid letters. If the original word based dictionary has 500k unique words, then the LTG representation has 10-fold reduction in dimensionality. However it is not easy to look up a word in such a mechanism. More storage may be used in order to facilitate the look-up of any LTG-word as we used in our work. Consider the following hash function of the LTG-word XYZ

$$h(xyz) = x(l+1)^2 + y(l+1) + z, \qquad (17)$$

where x, y, z are the numeric indices of X, Y, Z respectively and they are in the range from 0 to l. Consequently, the LTG-word XYZ corresponds to the $h(xyz)$-th word in the extended dictionary that has the size $(l+1)^3$. The additional space is due to those invalid LTG-words in the forms *#*, ##*, *##, and ###.

Besides the dimension reduction, there is another benefit of using the LTG based text representation: the morphological variants of a same text stream can be mapped to close vectors. This is encouraging since a query can always have misspelled forms. Take an example "banannna" vs. "bannana". They are two misspelled forms of the correct word "banana", and they have the same LTG words: #ba, ban, ana, nan, ann, nna, na#. While the correct spelling has the LTG words: #ba, ban, ana, nan, na#, with ana occurring twice, so the correct word has 5 LTG words with its two misspelled forms in common.

IV. Evaluation Metrics

We used two types of metrics to evaluate the model performance on the test set. The first is the average NDCG at a truncation level. Precisely, we define the average NDCG of the top i positions as

$$\overline{\text{NDCG}}_i = \frac{1}{N}\sum_q \left(\sum_{j=1}^{i} \frac{rel_j^q}{\log_2(1+j)}\right) \Big/ \left(\sum_{j=1}^{i} \frac{\overline{rel}_j^q}{\log_2(1+j)}\right), \qquad (18)$$

where $\overline{rel}_1^q \geq \overline{rel}_2^q \geq \cdots$ represent the descending order of $rel_1^q, rel_2^q, \ldots$, which are the relevance gains of the documents at positions 1,2, ... respectively under the query q. We require q to satisfy that $max_j rel_j^q - min_j rel_j^q \geq \epsilon > 0$, where ϵ is a pre-determined parameter. N is the total number of such queries in the test data set.

In the scenario where there is no preference on the order of the desired documents as long as they are returned among the top i positions, we use the second type of metrics. It's the average top-k ground truth labels' recall at the top i positions in prediction, where $k \leq i$. Precisely, it's defined as

$$\overline{\text{Recall}}_i^k = \frac{1}{N}\sum_q \frac{1}{k}\sum_{j=1}^{i} I(rel_j^q \geq \overline{rel}_k^q), \qquad (19)$$

where $I(\cdot)$ is the indicator function.

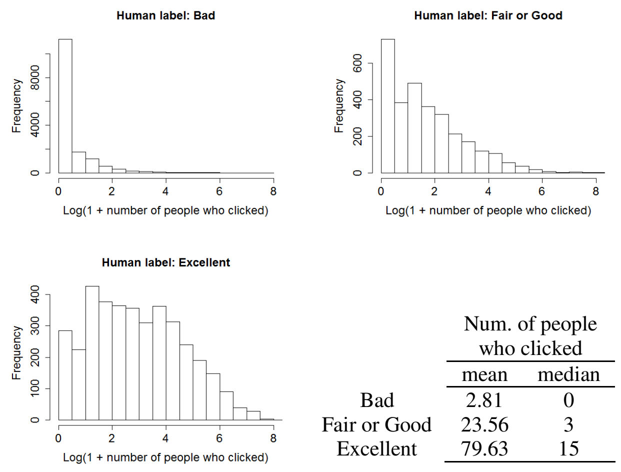
V. Experiments

In this section, we introduce the results of applying our model with the generalized loss function to the task of movie search. We collected the data from the Xbox One's query-logs, and used various algorithms and benchmarks including ours to predict the ranking order in a future period of time.

A. *Relevance Measure*

Many studies such as [5][6] have shown that the click-through data are effective in generating labels for learning ranking models. One relevance measure is the click-through rate (CTR). The CTR for a (q,d)-pair in a period of time is defined as the ratio of number of clicks to number of views. Although CTR is a good relevance measure by its definition, it's difficult to accurately calculate for our data. One reason is that it's hard to know whether the document d is viewed or not if it appears in a q-triggered session but is not clicked. Another reason is that one user might click d under q many more times than others do. In this case, the CTR calculation is biased toward this person. To avoid these issues, we use the number of distinct users who clicked d under q in a period of time as the relevance measure to determine the position of d in the ranking result of all the document candidates under q.

To show the validity of this measure, we sampled a set of 22,190 (query, movie)-pairs from the query-logs from December 2013 to March 2014, and obtained the 4-level human labels from 5 human judges. The four levels are *Excellent*, *Good*, *Fair*, and *Bad*. For each pair, we counted number of distinct users who clicked the movie under the query. The histograms of the logarithmic values under Bad, Fair or Good, and Excellent respectively are displayed in the following Fig. 2. It can be seen that the more people who clicked, the more relevant a document is under a query. Therefore, it's reasonable to treat more people who clicked as more relevant.



	Num. of people who clicked	
	mean	median
Bad	2.81	0
Fair or Good	23.56	3
Excellent	79.63	15

Fig. 2. The histograms of the logarithmic values of number of people who clicked, under different human labels.

Compared with the labels generated by the human judges, the labels decided by the number of people who clicked have some advantages. First, it's a good indicator of user engagement. For a popular query, it can reveal the intentions of different groups of people. The consensus is from a large number of real users other than a very limited number of human judges, therefore it contains much less bias. Second, human labels are expensive and it's very difficult to scale up, whereas the vast amount of click-through data can be obtained at very low cost.

Although the position bias (the higher the ranking position of the document shown to the user, the more likely it's clicked) is an important factor in typical web search problems, movie search has a quite different scenario due to its unique user interface. Usually, movie results are displayed in tile or icon layout styles that do not support the common top-down assumption of the web search. Moreover the picture of a movie's poster also affects its click probability. Therefore, click models that are sorely based on the analysis of position bias may not apply. On the other hand, number of people who clicked is an aggregated result, which is robust to noise. Empirically, at least for the movie search, we found the quality of this measure is comparable to human labels.

B. Data Preparation

We processed a set of query-logs from April 2014 to November 2014 and split it into the training part and the test part. The training part is from April 1, 2014 to September 30, 2014; the test part is from October 1, 2014 to November 30, 2014. For both the training part and the test part, for each (query, movie)-pair, we counted the number of distinct users who clicked the movie under the query and used it as the label.

Previous studies such as [6] have shown it is important to remove noise from the click-through data, therefore we set a threshold to filter out spam queries. Precisely, a query is viewed as a spam query if all the movies under it were clicked by at most 1 distinct user. In other words, we only kept the queries under each of which there is at least one movie that was clicked by at least 2 distinct users. We did this filtering for both the training part and the test part. Additionally, for the test set, we increased the threshold by 1 and removed any query with only identical labels since it is impossible to evaluate the performance difference in this case.

After the filtering, there are 674,307 unique (query, movie)-pairs in the training set, with 26,958 unique queries; and there are 176,181 unique (query, movie)-pairs in the test set, with 7,018 unique queries. In the training set, there are 106,285 clicked (query, movie)-pairs, and in the test set, there are 27,595 clicked pairs.

The average query length is 2.40 for the training set and 2.30 for the test set. The document of a movie contains 5 fields: release date, title, actors, genre, and region. To build the data for training and testing DSSM and our model, we form the text string of a document via the concatenation as: release date + title + actors + genre + region.

There are 47,069 unique movies in the processed training and test sets. The average document length is 18.36 for the training set and 20.48 for the test set. The following Table I summarizes the basic statistics.

TABLE I. DATA STATISTICS

	Training set	Test set
Time window	2014-06-01 to 2014-09-30	2014-10-01 to 2014-11-30
Num. of unique queries	26,958	7,018
Num. of unique pairs	674,307	176,181
Num. of unique clicked pairs	106,285	27,595
Ave. query length	2.40	2.30
Ave. doc. length	18.36	20.48

The labels in both the training and the test sets have long tail distributions. The following Fig. 3 shows the histograms of the logarithmic values of the labels in two scenarios for both the training and the test sets. In scenario 1, the histogram is generated from all (q, d)-pairs (clicked or not clicked). The scenario 2 is the scenario 1 zoomed in on the clicked (q, d)-pairs only. The zoomed-in histograms indicate that the majority of the clicked pairs have labels 1 to 3.

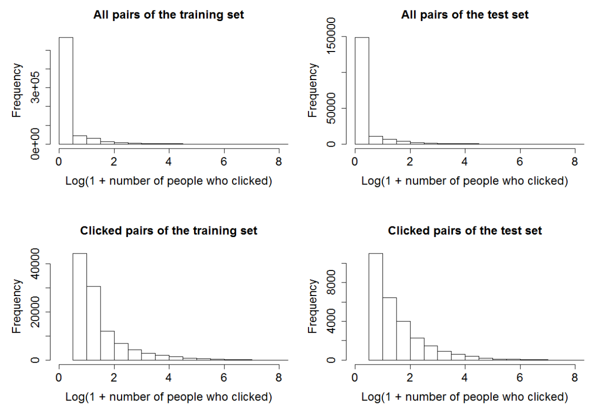


Fig. 3. The histograms of the logarithmic values of number of people who clicked for scenario 1 (all pairs) and scenario 2 (clicked pairs only) in both the training and the test sets.

Among the 22,190 (query, movie)-pairs from the query-logs from December 2013 to March 2014 that have 4-level human labels (Excellent, Good, Fair, Bad), there are 3,763 labeled Excellent, 3,016 labeled Good or Fair, and 15,411 labeled Bad. There are 3478, 2287, and 4195 that were clicked in the three groups, respectively. Therefore, the likelihoods of being clicked for the three groups are 0.924, 0.758, and 0.272, respectively. We used that information to fit the parameters of the following label mapping function

$$y(s) = \frac{1}{1+\exp(-\sigma(s-a))} \tag{20}$$

and we found $\sigma = 0.2641$ and $a = 1.2402$. The plot of this function is shown as the following Fig. 4.

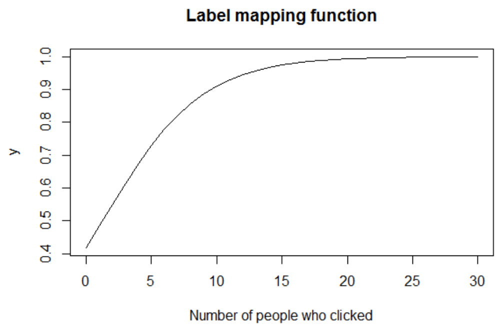


Fig. 4. The plot of the label mapping function.

This label mapping function was used to transform the raw labels of the training set into values between 0 and 1 to approximate the true probabilistic target so that the generalized loss function L_2 in (2) can be constructed from the training set. Note that we don't transform the raw labels of the test set since we use the raw labels in the test set for evaluation. Consequently, the pairs in the training set for our model have the label gains as the mapped values and the pairs in the test set for all methods have the label gains as the original numbers of people who clicked.

C. *Model Setting*

As in the existing state-of-the-art work in [13], for both the embedding functions u and v, we adopted the neural network structure that is illustrated in the following Fig. 5.

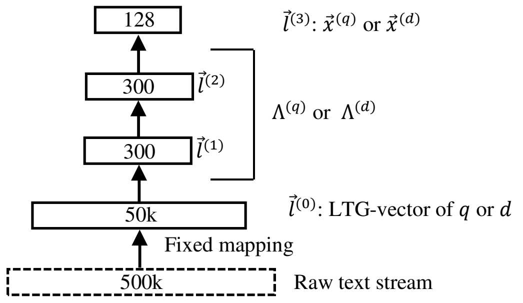

128
$\vec{l}^{(3)}$: $\vec{x}^{(q)}$ or $\vec{x}^{(d)}$
300
$\vec{l}^{(2)}$
$\Lambda^{(q)}$ or $\Lambda^{(d)}$
300
$\vec{l}^{(1)}$
50k
$\vec{l}^{(0)}$: LTG-vector of q or d
Fixed mapping
500k
Raw text stream

Fig. 5. The neural network structure of the embedding functions. There are four layers. The input layer corresponds to the LTG-vector representation of the raw text stream. The output layer corresponds to the vector in the semantic space. There are two intermediate layers of dimension 300.

Note that the input layer is the LTG based vector representation. The mapping from the raw input text steam to the LTG-vector is found by hashing and is fixed throughout the model training. The model was trained by using the mini-batch version of the stochastic gradient descent method [3]. Each mini-batch consists of 1024 randomly selected training instances. The learning rate is adaptive with initial value 0.5. For the softmax function, we set $\gamma = 10$ and $J = 4$.

D. *Comparison Setting*

We compared our model with the generalized loss function L_2 in (2) against two sets of baseline models. The first baseline is DSSM with the loss function L_1 in (1). Since DSSM with the loss function L_1 has already been compared to a lot of benchmarks, we provided additional benchmarks for comparison as the second set of baselines: the first one is the BM25F [18], which is an unsupervised learning algorithm; the second one is BLTM[9], an extension of LDA which can calculate the query document similarities at the topic level; and the third one is LambdaMart, which is a widely used supervised learning algorithm and it generates a model in a form of gradient boosted decision trees. To use the LambdaMart algorithm, we manually generated about 2,000 term or lexical matching features.

To be consistent with the existing state-of-the-art work such as [13][20], all models in this study are trained from the clicked pairs in the training set. Since there are only 106,285 clicked pairs in the training set, for training DSSM and our model, we took the model produced by Huang, He, Gao and Deng [13] as the seed one and tuned its parameters using the 106,285 clicked pairs of the training data. The seed model has the same neural network structure and was trained from the clicked pairs of a large set of query logs of the Bing search. The loss function for training the seed model is the same as L_1 in (1), and the seed model is denoted as *Seed_DSSM*. The DSSM model (denoted as *DSSM*) and our model with generalized loss function (denoted as *GDSSM*) were obtained by tuning the seed model under the loss functions L_1 in (1) and L_2 in (2), respectively.

For the LambdaMart algorithm, we designed two versions of experiments depending on what features are used. One version only uses the manually generated term or lexical match features, and it's denoted as *LM_base*. The other version uses both the term or lexical match features and the semantic similarity feature generated by our model, and it's denoted as *LM_GDSSM*. It's very interesting to see whether there is significant performance lift if the semantic similarity score computed from our new model is added as a new feature. To train a model using the LambdaMart algorithm, we used the raw labels other than mapped labels since the LabmdaMart can take integers as target labels.

Each trained model is called a ranker in this paper. All of BM25F, DSSM and GDSSM served as single feature rankers since their values directly decide the ranking order, whereas the counterparts of LambdaMart models LM_base and LM_GDSSM are referred as multi-feature rankers since they combine multiple features. After using each model to score the (query, movie)-pairs in the test set, we calculated $\overline{\mathrm{NDCG}}_i$ (average NDCG of the top i positions) for $i = 1,3,10$ and $\overline{\mathrm{Recall}}_i^3$ (average top 3's recall at the top i positions) for $i = 3,6,10$. In the end, the recentness is an important factor to decide the appropriate ranking order of movies, therefore we also built simple linear regression models to combine the rankers' prediction with the recentness signal to see if we can further improve the performance.

E. *Results*

The test results were summarized in the following Tables II-V. From the tests for single feature rankers, it is shown that GDSSM does have superior performance over DSSM, Seed_DSSM, BLTM, and BMF25 with respect to both the NDCG and the recall metrics. It's interesting to observe that the overall order of the single feature rankers' NDCG and recall performance is GDSSM > DSSM > BLTM > BMF25 > Seed_DSSM. The reason why Seed_DSSM is the worst is because it was trained from the context of web search, while the other four were trained from the specific context of movie search. The fact that GDSSM is significantly better than DSSM shows that fine-grained relevance label structure is very helpful for capturing the subtle relevance differences between various documents under the same query, which in turn leads to the performance improvement.

Regarding the multi-feature rankers, LM_GDSSM is significantly better than LM_base in both the NDCG and the recall values. That is to say, adding the score computed from GDSSM as a semantic similarity feature to LambdaMart that only uses term or lexical match features can boost the performance.

We can see that the multi-feature ranker LM_base achieves better NDCG values than single feature ranker GDSSM (please refer to Tables II and IV) but GDSSM has better recall values (please refer to Tables III and V). The main reason why LM_base beats GDSSM in NDCG measures is that the LambdaMart was designed for optimizing NDCG directly [4] and the NDCG measurement emphasizes the top few results, while our model and DSSM optimize the similarity between the query and document in a semantic space but the relative order of the documents under the same query is not directly reflected in the objective functions.

The observation that GDSSM has better recall values compared to LM_base implies that term or lexical matching based retrieval could miss important semantically matched contents. Therefore it is not surprising to see LM_GDSSM that combines both the term or lexical matches and the semantic matches yields further performance lift.

TABLE II. THE SINGLE-FEATURE RANKERS' NDCG PERFORMANCE

	Without recentness adjust			**With recentness adjust**		
Average NDCG@i	i=1	i = 3	i = 10	i = 1	i = 3	i = 10
BM25F	0.6296	0.7116	0.7718	0.5740	0.6805	0.7465
BLTM	0.5439	0.6821	0.7542	0.5801	0.7021	0.7687
Seed_DSSM	0.4502	0.6000	0.6887	0.4435	0.5977	0.6861
DSSM	0.5875	0.7186	0.7820	0.6127	0.7386	0.7977
GDSSM	0.7265	0.8107	0.8528	0.7508	0.8260	0.8657

TABLE III. THE SINGLE-FEATURE RANKERS' RECALL PERFORMANCE

	Without recentness adjust			**With recentness adjust**		
Average top 3 recall@i	i = 3	i = 6	i = 10	i = 3	i = 6	i = 10
BM25F	0.5589	0.7603	0.8420	0.5620	0.7657	0.8476
BLTM	0.5643	0.7702	0.8512	0.5747	0.7809	0.8625
Seed_DSSM	0.5285	0.7491	0.8371	0.5375	0.7575	0.8448
DSSM	0.5910	0.7996	0.8771	0.6001	0.8077	0.8837
GDSSM	0.6243	0.8235	0.8899	0.6322	0.8271	0.8950

TABLE IV. THE MULTI-FEATURE RANKERS' NDCG PERFORMANCE

	Without recentness adjust			**With recentness adjust**		
Average NDCG@i	i = 1	i = 3	i = 10	i = 1	i = 3	i = 10
LM_base	0.7921	0.8263	0.8638	0.8006	0.8346	0.8719
LM_GDSSM	0.8261	0.8647	0.8957	0.8285	0.8685	0.8983
Improvement	4.29%	4.65%	3.69%	3.49%	4.06%	3.03%

TABLE V. THE MULTI-FEATURE RANKERS' RECALL PERFORMANCE

	Without recentness adjust			**With recentness adjust**		
Average top 3 recall@i	i = 3	i = 6	i = 10	i = 3	i = 6	i = 10
LM_base	0.6115	0.7963	0.8697	0.6165	0.8071	0.8786
LM_GDSSM	0.6430	0.8346	0.8965	0.6448	0.8343	0.8989
Improvement	5.16%	4.80%	3.08%	4.60%	3.37%	2.31%

VI. CONCLUSION

In this paper, we have introduced the generalized loss function L_2 in (2) for the semantic similarity models that have neural network structures. It's motivated by the fact that for data with a fine-grained target structure, it's possible to build better labels to improve the prediction. We analyzed the generalized loss function and pointed out that label improvement can make considerable contribution toward reducing the discrepancy between the prediction and the true target. We trained models and performed extensive experiments using the Xbox One's logs on movie search. We found evidence that the generalized loss function is significantly better than the original loss function, and other benchmarks, measured by the NDCG and recall metrics.

We also compared our model GDSSM with the generalized loss function L_2 against the current widely used search ranking algorithm LambdaMart that uses thousands of manually generated term or lexical match features and found that our model has better recall performance. Moreover, by adding the similarity score computed from our model to the LambdaMart as a semantic match feature, significant performance lift is achieved in both NDCG and recall measurements. These results are encouraging since they indicate some progress in saving the engineering efforts of manually building features.

As for future work, we suggest adding more structure(s) to the architecture of our model GDSSM to enrich the feature generation from the input layer. It will be very interesting to compare these new type of models to the LambdaMart that uses both the term or lexical matching features and the semantic matching features.

REFERENCES

[1] Y. Bengio. Learning Deep Architectures for AI. In Foundations and Trends in Machine Learning, vol. 2, pp. 1-127, 2009.

[2] D. M. Blei, A. Y. Ng and M. J. Jordan . Latent Dirichlet Allocation. In JMLR, vol. 3, 2003.

[3] L. Bottou. Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010, pp. 177-186, 2010.

[4] C. Burges. From RankNet to LambdaMart to LambdaMART: An Overview. Technical Report, No. MSR-TR-2010-82, 2010.

[5] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu and P. P. Kuksa. Natural Language Processing (Almost) from Scratch. In JMLR, vol. 12, pp. 2493-2537, 2011.

[6] Z. Dou, R. Song, X. Yuan and J. Wen. Are click-through data adequate for learning web search rankings? In CIKM, pp. 73-82, 2008.

[7] S. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas and R. A. Harshman. Indexing by Latent Semantic Analysis. In J. American Society for Information Science, 41(6): 391-407, 1990.

[8] S. T. Dumais, T. A.Letsche, M. L. Littman and T. K. Landauer. Automatic Cross-linguistic Information Retrieval Using Latent Semantic Indexing. In AAAI-97 Spring Symposium Series: Cross-Language Text and Speech Retrieval, 1997.

[9] J. Gao, K. Toutanova and W. Yih. Clickthrough-based Latent Semantic Models for Web Search. In SIGIR, pp. 675-684, 2011.

[10] J. Gao, X. He, W. Yih and L. Deng. Learning Continuous Phrase Representations for Translation Modeling. In ACL, pp. 699-709, 2014.

[11] M. Girolami and A. Kaban. On an equivalence between PLSA and LDA. In SIGIR, pp. 433-434, 2003.

[12] T. Hofmann. Probabilistic latent semantic indexing. In SIGIR, pp. 50-57, 1999.

[13] P. Huang, X. He, J. Gao, L. Deng, A. Acero and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In CIKM, pp. 2333-2338, 2013.

[14] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In SIGIR, pp. 41-48, 2000.

[15] S. Kullback and R.A. Leibler. On Information and Sufficiency. Annals of Mathematical Statistics 22 (1): pp. 79–86, 1951.

[16] G. Mesnil, X. He, L. Deng and Y. Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In INTERSPEECH, pp. 3771-3775, 2013.

[17] J. Platt, K. Toutanova and W. Yih. Translingual Document Representations from Discriminative Projections. In EMNLP, pp. 251-261, 2010.

[18] S. E. Robertson and H. Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. In Foundations and Trends in Information Retrieval, 3(4): 333-389, 2009.

[19] R. Salakhutdinov and G. Hinton. Semantic Hashing. In Proc. SIGIR Workshop Information Retrieval and Applications of Graphical Models, 2007.

[20] Y. Shen, X. He, J. Gao, L. Deng and G. Mesnil. A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval. In CIKM, pp. 101-110, 2014.

[21] X. Song, X. He, J. Gao and L. Deng. Unsupervised Learning of Word Semantic Embedding using the Deep Structured Semantic Model. Technical Report, No. MSR-TR-2014-109, 2014.

[22] W. Yih, X. He and C. Meek. Semantic Parsing for Single-Relation Question Answering. In ACL, pp. 643-648, 2014.