Learning to Rank-based Approach for Movie Search by Keyword Query and Example Query

Kosuke Kurihara*
Yahoo JAPAN Corporation
Tokyo, Japan
kurihara@sw.it.aoyama.ac.jp

Yoshiyuki Shoji
Aoyama Gakuin University
Kanagawa, Japan
shoji@it.aoyama.ac.jp

Sumio Fujita
Yahoo JAPAN Corporation
Tokyo, Japan
sufujita@yahoo-corp.jp

Martin J. Dürst
Aoyama Gakuin University
Kanagawa, Japan
duerst@it.aoyama.ac.jp

ABSTRACT

This paper proposes a method for ranking movies by keyword queries and examples using machine learning techniques that analyze actual data from the online movie review site. Existing search methods cannot rank movies in "surprising order" for the keyword query "surprising." People and critics created many "My best surprising movies" rankings on the web. Our proposed method uses a LambdaMART, one of the mainstream Learning to Rank techniques, to learn these personalized rankings and sort the movies through the viewpoint represented by a given query. To accept more complex information needs, we diverted the learning results to a search-by-example algorithm that enables users to input examples, such as "surprising movies like *The Usual Suspects* or *Fight Club*." The experiment using the personal ranking data from the personal content curation service in Yahoo! Movies Japan suggests two findings: direct learning of personal ranking does not improve search performance, and the search-by-example-based application increases user satisfaction.

KEYWORDS

Learning to Rank, Online Review, Movie Search

1 INTRODUCTION

Finding a movie to watch is a complicated technological challenge in information retrieval. Imagine that a user is looking for a movie through a certain keyword, such as "surprising movie." Since a movie is a visual image, it is impossible to search its content directly by keywords. Additionally, metadata and synopses cannot be used to rank movies correctly. The words related to "surprising" are not included in the synopsis of a surprising movie.

Under these conditions, movie reviews are an essential source of information for selecting a movie to watch. Currently, people routinely read reviews on online review sites or video streaming services. Users can read through movie reviews to discover the reputation of a certain movie. It means that users can decide whether the movie is surprising. However, they cannot search for movies by entering keywords and find movies relevant to the keyword. In other words, it cannot answer the request for a list of surprising movies.

There is no established method for integrating many review information and connecting the reputation of each movie with the query entered by the searcher to enable movie ranking. Many movie review sites provide simple movie rankings, such as the average score ranking for each movie category. However, users' search intents are not limited to categories only but are more diverse. For example, there is a need to rank movies from various viewpoints, such as "road movies that make me want to go on a trip when I saw them" or "movies suitable for watching with my girlfriend." Many articles based on individual viewpoints, such as "My top ten sad movies" or "Ten best movies that will inspire you to travel," are published on the website, attracting many readers. It is also common for individuals to create such rankings. Many online curation sites allow users to create rankings, such as "My ten best surprising movies" from the viewpoints that they are interested in.

Current movie information websites cannot automatically generate such rankings as "in order of tear-jerker" or "in order of surprise" for any given query. Therefore, if a user wants to find "the tear-jerker movie to watch next," they have to read the reviews posted for each movie one by one to determine whether they match their information needs. Such activity is a time-consuming process and involves the risk of seeing spoilers that they do not want to know before they watch the movie.

Therefore, in this study, we propose a search model that can rank movies based on a given viewpoint. For this purpose, our method considers two improvements:

- Learning to Rank (LtR) technique is used to learn many movie rankings created by individuals;
- Introducing the concept of the search-by-example approach to the movie search

for traditional similarity-based methods.

We have previously developed an algorithm that makes distributed representation of words in movie reviews and enabled collecting reviews containing words related to a given keyword query. In this study, we aggregate such words and rank movies using LtR techniques. We collect private movie rankings, such as "My Best Ten *** Movies," curated by individuals and posted on actual movie information sites. Then, using a LambdaMART, one of the most widely used LtR techniques, we trained the model on the collected data. This model allows users to search for movies

*Kosuke Kurihara contributed to this research while at Aoyama Gakuin University until March 2021.

by arbitrarily keyword query, such as "tear-jerker" or a movie that "makes me hungry."

Finally, we introduce the idea of search-by-example to movie search to accept information needs with a finer granularity than keyword queries. Keyword queries often fail to represent the search intent sufficiently. For instance, movies, which match the keyword query "nostalgic" should be divided into two types. One is the movies whose topics are nostalgic (*e.g.*, a drama set in the 1980s, released in 2021), and the others are memorable to individuals (*e.g.*, a drama set in the future, released in the 1980s when the searcher was a child). To solve this problem, we created a search algorithm that finds movies that can fill in the blank of the example ranking given by a user. Users can input their information needs as an example, such as "surprising movies like *The Usual Suspects* or *Fight Club*." It should make it easier to find the movie users want instead of inputting "surprising."

These two improvements should make users search for movies based on any viewpoint. We conducted experiments on a large dataset of actual services to verify the effectiveness of these methods. For the experiment, we used actual data from the "Roundup [1]" service of Yahoo! Movies, Japan. This service allows users to list up to ten movies as they want and save or publish them. The data of this service include many users' original rankings, as the official support recommends, "Let's make your original movie ranking[2]." We conducted two experiments for each improvement: a large-scale crowdsourcing-based labeling evaluation and a user experiment. In the large-scale crowdsourcing experiment, users labeled the movies in search result rankings for a pre-prepared query to check whether they matched the query. In the subject experiment, participants used our search system based on search-by-example. Then, they were asked whether they were comfortable with this type of search model in a questionnaire.

The remainder of the paper is organized as follows. This section described the social and technical background and an overview of this study. Section 2 introduces the related work. In Section 3, we describe the details of the proposed algorithm. Sections 4 and 5 describe the evaluation experimental methods and results for our ranking algorithm and search-by-example application, respectively. Section 6 discusses the results. Finally, Section 7 presents the conclusion.

2 RELATED WORK

This study focuses on information retrieval using user reviews, reputation information on the web, and LtR techniques. In this section, we introduce and discuss the existing mainstream methods of using reputation information on the web, such as distributed representation-based item search and recommendation, LtR, and search-by-example model.

2.1 Item Search Using Embedding Methods

Our method ranks the items (*i.e.*, movie) by creating a distributed representation using their review text. Using the embedding-based approach, such as the distributed representation of entities, has become one of the mainstream methods for search and recommend items currently.

Barkan *et al.*,[1] proposed an item embedding method called Item2Vec. Item2Vec applied the skip-gram model used in Word2Vec to vectorize items on an e-commerce site. It uses the set of items purchased simultaneously as contextual information to represent an item. Additionally, it assumes that the feature of the purchased item is represented by the items purchased before and after it (*i.e.*, the skip-gram model). This distributed representation of items can be used for information recommendations, such as collaborative filtering. Yao *et al.*,[15] compared multiple-item embedding methods in an information recommendation task. They used a movie dataset for comparison. In addition to Item2Vec, the comparison methods include Doc2Vec for simple reviews. Many other methods vectorized items using the 2Vec-based approach to enable the search or recommend them. For instance, Product2Vec by Chen *et al.*,[3] similarly analyzed purchasing behavior by creating a distributed representation of items. Similar to our study, Zhang *et al.*,[17] proposed Movie2Vec, which extends the idea of transformers used in BERT to movies and their viewing data. Thus, the distributed representation of movies can be used for a cold-start item recommendation, such as item-based recommendation.

In this study, we also treat movies as vectors. Many previous studies have ranked vectors of items by simple similarity calculations (*i.e.*, cosine similarity or Euclidean distance). We use the movie's distributed representation as an input of LtR. For this purpose, we used target topic aware Doc2Vec (TTA-D2V), developed in our previous study, to vectorize movie reviews and aggregate them for each movie.

2.2 Item Search Using Learning to Rank

LtR is a method for solving ranking problems through supervised learning methods. The dataset of LtR consists of queries and documents with correct answer labels. The dataset S is given by

$$S = \{(q, i, r) \mid (q, i) \in P, r = relevance(q, i)\}, \tag{1}$$

where (q, i) is a tuple of a query q and the item i, and P is a set of all tuples, r is a label of relevance between the query q and the item i. The model predicts the degree of relevance r from a tuple (q, i). It is widely used as an optimization method, especially for web search systems since it can infer the order of itemsets whose order is unknown.

Karmaker *et al.,* [5] investigated the usefulness of LtR for product search in e-commerce sites by testing several hypotheses. LtR is more accurate than logistic regression and SVM as a product search method. This trend is true, especially when the model is trained to predict the purchase and click rates of products.

Ludewig *et al.,* [10] propose a hotel-ranking method using LtR. They provided highly accurate hotel rankings by deriving users' preferences and search intentions from short-term interactions without using their long-term preference profile. Studies on item retrieval based on deep learning have also been conducted [9].

Most existing studies in the information retrieval field using LtR used click logs or purchase histories as the correct training data [13, 14]. However, in this study, we use many rankings created by users as training data. We believe that this can optimize the

[1]Yahoo! Japan: Yahoo! movie - My movie (in Japanese) https://movies.yahoo.co.jp/my/matome/

[2]Yahoo! Japan: How to use Roundup function (in Japanese) https://support.yahoo-net.jp/PccMovies/s/article/H000011620

rankings without using difficult-to-collect data, such as click logs or purchase histories.

2.3 Search-by-Example

Search-by-example is a method of input user information needs for the search engine. It has been classically proposed as one of the database query languages. Additionally, it can be used when it is not easy to express the search intent as a keyword.

This idea is popular in the music retrieval field [7, 11, 12]. When searching for music without lyrics, the user cannot search for it with a keyword-based search system. As a classic example, Ghias *et al.,* proposed a method for retrieving music by humming (Query by Humming).

As an example of using outside music, Kato *et al.,* [6] proposed a method for finding facilities in unknown areas by entering examples in areas where the user is familiar with. Similarly, there are cases of searching for short videos that include human actions as something difficult to enter directly with keywords [16].

We believe that movie search is also a domain where it is not easy to express search intents directly in keyword queries. When looking for a movie that no user has seen yet, it is not easy to describe its content in a keyword query. The proposed method finds movies similar to the movie by giving several examples of movies that contain similar viewpoints.

3 METHOD

We propose two movie search algorithms based on LtR: keyword search and search-by-example methods. The two proposed methods use the review information to vectorize movies and employ a machine learning method called LambdaMART to rank movies.

3.1 Method Overview

These two search methods share several similar parts. The framework, vectorization of movies, and re-ranking search results by LambdaMART [2] are common to both algorithms.

An overview of the shared part of the algorithm is as follows:

Algorithm 1 Create Movie Ranking

Require: q:Query
Ensure: R:Movie Ranking (list)
  function GENERATE_RANK(q)
    $list$:= GET_MATCHED_MOVIES(q)
    R := RERANK($list$)
    return R
  end function

This algorithm performs based on the re-ranking task in LtR. First, the algorithm takes the set of items $list$ that is the top n items in the ranking sorted by the simple topic relevance. Next, it sorts the set of items $list$ using a trained model and creates the final ranking R.

To obtain a list of movies to be re-ranked, we vectorized each movie and keyword query using TTA-D2V. In 2019, we proposed the TTA-D2V method that can correctly vectorize short reviews by learning Doc2Vec with a focus on other reviewers' reviews that mention the same movie (See previous work for details [8]). Then, we used the top 1,000 movies and keyword queries as the target of the re-ranking.

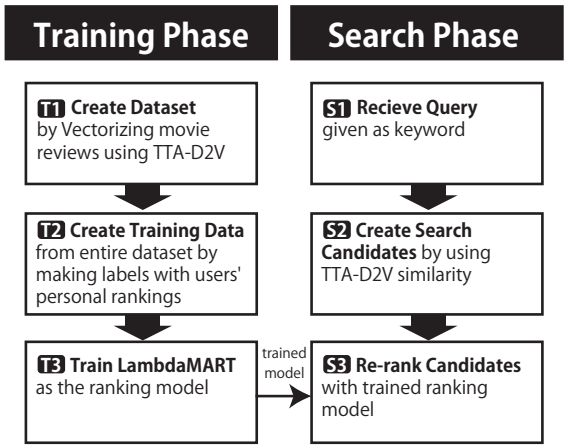

Figure 1: Overview of keyword query search system using user personal movie rankings and LambdaMART.

The framework that extracts such candidate search results and re-ranks them is the same; however, the two methods' timing and training data are different. In keyword query search, personal rankings posted by many users are used as the training data. It calculates the model in advance since it uses massive training data. In the search-by-example model, the method uses examples of movies received from users as training data. Each time the user enters a list of example movies as a query, the model will re-learn the ranking.

3.2 Keyword Search by Learning to Rank

The keyword search method receives keywords from the user and returns a ranking. This method aggregates many people's opinions: the individual rankings written by many reviewers are integrated and used to rank movies by the viewpoint given as a query.

The generated ranking will sort movies suitable for the viewpoint on the movie expressed by the input query, in the order of the depth of the degree of that viewpoint. For example, in the case of a search about "tear-jerker movie," the system sorts the movies in the order that they make people cry more by integrating personal rankings, such as "Ten of my saddest movies" created by reviewers. The method enables searchers to easily search for movies using free keywords, such as "movies that inspire me to travel" or "movies suitable for dating."

Figure 1 shows an overview of the keyword search method. The method can be divided into two phases: the preparation and retrieval phases. The preparation phase is performed only once at first. The search phase is performed after completing the preparation phase and repeated for each search.

3.2.1 Training Phase. The preparation phase consists of three steps. The goal of the preparation phase is to learn a single ranking model by integrating many fragmented rankings.

Table 1: Feature vector of the entity in a private ranking

Feature	# dimensions
Movie metadata	23
Movie Content (Distributed Expression of the review)	300
Query (Distributed Expression of the ranking title)	300
Relevance of Query and Movie (Cosine simirarity)	1
Total	624
Label (inverted rank)	1

In the first step (T1 in Figure 1), the method creates a distributed representation of each movie using reviews. For the vectorization of movies, the method trains the TTA-D2V model in advance. TTA-D2V is a modified algorithm of Doc2Vec that trains a model to estimate which movie a single sentence in a review was written. The trained model can create an arbitrary dimensional vector suited for movie reviews from any sentence or word. Then, the method considers all reviews written for a movie as a single document and vectorizes it for each movie.

In the second step (T2 in Figure 1), the method creates training data from the entire dataset. LamdaMART requires training data as a set of three-tuple consisting of a query, relevance score, and feature vector. Since our data is a personal ranking, we used the ranking title as the query. We used language patterns to extract the query part from their title since the ranking title is a concise sentence. Specifically, we removed parts of the title, such as "My best" or "Top ten," and used nouns and adjectives preceding them as the query. As a relevance score, we use the inverse of the rank within the personal ranking.

The feature vectors of the entities in the ranking consist of four types of features: the movie's metadata, review text, query, and the relationship between the query and the movie. Table 1 presents the details of the features. As the metadata of a movie, we use 23 dimensions to represent the average rating of the movie on movie review sites, the number of users who have registered the movie to the watch list, and the category tag of the movie. As the review contents, we use the distributed representation of the movie as vectors of 300 dimensions each. As a query, we also used the 300-dimensional distributed representation of the query. By treating the query as a distributed representation, we can handle cases where a query that is not included in the training data is input during the search. As the relationship between the movie and the query, their distributed representations' cosine similarity is also used as a one-dimensional feature.

The third step is to train the model (T3 in Figure 1). The model is then created by training the prepared training data. Negative examples are necessary for training. We select five movies that are not similar to the query. We vectorize and use them as negative examples. We trained the LamdaMART model with one dataset consisting of positive examples (*i.e.*, movies included in the ranking submitted by users) and five negative examples. The preparation phase is completed by learning all the rankings.

3.2.2 Search Phase. The search phase consists of three simple steps: receiving a keyword query, collecting search suggestions, and re-ranking.

First, the system receives the user's keyword query consisting of a few words. Since the keyword query is used for similarity calculation using a distributed representation, it needs to be preprocessed. As a preprocessing, the system removes natural language parts in the query by language patterns (e.g., "movie that," "a film of"). Then, the system performs morphological analysis to extract only words, such as nouns, verbs, and adjectives. It also removes stop words and words that do not appear in the dataset. The preprocessing was necessary to unify the vocabulary used in reviews and the query since it uses the Doc2Vec model trained on the movie review dataset.

Next, the method uses the TTA-D2V model learned in the training phase to represent the query as a distributed representation. It calculates the similarity between the distributed representation of the query and all movies in our dataset to extract the candidates. Their cosine similarities are calculated, and the top 300 movies are targeted for re-ranking.

Finally, the ranking model learned in the preparation phase is used to sort the candidates. The method finally returns these re-ranked candidates as the search result. The search results ranking should be an aggregation of many users' rankings based on the viewpoint given by the query.

3.3 Search-by-Example

The search-by-example method accepts as a query an example consisting of several movies created by the user. This method allows for more detailed input of search intent. In other words, instead of saying "science fiction movies," the system must be able to guess the user's intent more clearly if users can input search requests as "science fiction movies like *Blade Runner* or *2001: A Space Odyssey*." The input and output of this method is a movie ranking. The method learns the input movie ranking, inserts new movies at the end or in the ranking gaps.

Figure 2 shows the overview of the search-by-example method. Similar to the keyword search method described above, the search-by-example method also consists of two phases: preparing and search phases. The most significant difference from the keyword search method is that it does not perform large-scale learning in the preparation phase but performs small-scale learning in the search phase each time. It means that the functions related to training have been moved from the training phase to the search phase, but the content of each step has not changed. The keyword search algorithm learned the model from the large-scale rankings made by many people, registered on the Roundup service. However, the search-by-example algorithm learns search models by on-the-fly learning using a single ranking entered by the user on-site.

The method executes the preparation only once to enable movie vectorization by learning the movie reviews. This phase consists of only one step. The method vectorizes all movies in advance using reviews to determine the characteristics of movies in the given example ranking. We used TTA-D2V to vectorize the movies from

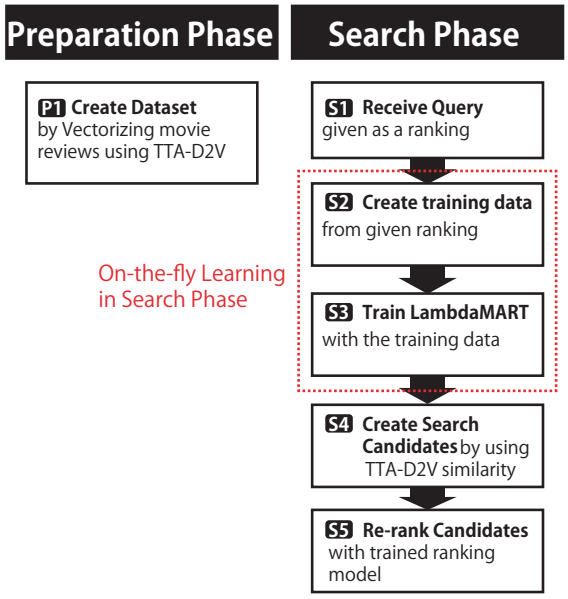

Figure 2: Overview of search-by-example-based movie search model

Table 2: Features of a movie in search-by-example model

Feature	# dimensions
Movie metadata	23
Movie Content	300
Total	323

the reviews similar to keyword search. The details are the same as those described in section 3.2.1.

Next, the method executes the search phase, and the search phase is required every time for every search.

The method accepts an example movie ranking given as a query. The ranking should be based on some uniform viewpoint, such as "Top ten movies I feel nostalgic about" or "Movies that made me hungry while watching." An example ranking should consist of about ten movies.

Then, the method generates the training data for LtR on the fly. Unlike the keyword search method, only the example movie ranking given by the searcher is used as training data. In this way, the ranking model obtained because of training depends on the ranking given by the individual searcher.

For the actual training, the method represents the movies in the ranking as features. The features used are shown in Table 2. The metadata and distributed representation of the movies are the same as in the keyword search method. Unlike in keyword search, we do not use query features.

We used the generated training data for training to create the LambdaMART-ranking model. Since negative examples are necessary for the training of the lambdaMART model, the method selects movies that are not similar to the input movies. The method vectorized the movies in the input ranking and calculates their arithmetic mean to obtain a set of roughly dissimilar movies. Then, we selected five movies with the lowest cosine similarity to the arithmetic mean vector from all movies vectorized in the preparation phase and used as negative examples.

The system determines the candidate movies to create the search result. Like the keyword search model, this model creates the final search result by re-ranking candidates. Therefore, the method selects a set of roughly similar movies to the input movies and sorts them into an order consistent with the input ranking. The method vectorized each movie in the example ranking given as a query. Then, the method selects the movie with the top n highest cosine similarity to each movie in the example ranking. The number of candidates n should be defined by considering the computational cost, the number of movies given as an example ranking, and the number of search results. In our experiment, we used 30 movies close to each movie in ten input movie sets for re-ranking, creating a ranking of 30 movies. It is worth noting that, in many cases, a set of movies with consistent concepts may be placed close together in the vector space. When the input is a ranking consisting of movies sharing a certain perspective, the candidate movies that are similar to them often overlap. Here, the number of similar movies to be included in the candidate list will be dynamically adjusted according to the number of search results required.

Finally, the actual re-ranking is conducted to create the search results. The ranking model learned from the input example ranking re-ranks the candidates. The movies ranked in the ranking are presented as a search result.

4 CROWDSOURCING-BASED EXPERIMENT FOR KEYWORD SEARCH

To quantitatively verify the ranking accuracy of the keyword search method, we conducted a large-scale evaluation experiment through crowdsourcing using real data.

To verify the usefulness of LtR using data from Roundup sites, we compared a model with simple relevance and a model that re-ranks with LtR. We have prepared two ranking algorithms. **LtR** is a proposed method described in Section 3.2. It re-ranks the top 300 candidates collected by their relevance using lambdaMART. **TTA-D2V** is a comparative method without re-ranking the candidate. It simply ranks movies by cosine similarity of their TTA-D2V feature vector.

We searched for movies using ten pre-prepared queries and created rankings for each method. Each movie included in the ranking was randomly sorted, and a questionnaire was administered. Participants judged how well a movie matched the query, e.g., "On a scale of one to four, how much does Titanic make you cry?"

The questionnaire was administered through crowdsourcing. As a crowdsourcing platform, we used Yahoo! Crowdsourcing, a popular platform in Japan. A total of 1,024 workers engaged in 3200 evaluation tasks.

4.1 Implementation

To verify the performance of the proposed method, we implemented a web system that collects data from the Roundup service of an

actual site and trains the ranking model from it. In this section, we describe the details of the actual dataset and the system implemented.

We used data collected from Yahoo! Movies in Japan as a movie dataset. Yahoo! Movie is a portal site for movie information. Therefore, it is possible to collect metadata for movies and reviews for them together. Additionally, this site allows users to create their rankings in the Roundup service. Users can create as many movie rankings as they want with any title, consisting of up to ten movies.

From the Roundup data, we extract the actual rankings for learning. The data are not clean and need cleansing since they are prepared by many users. Many rankings created by individuals do not have a specific viewpoint. For example, rankings, such as "my top ten favorite movies" or "top ten movies of 2020" do not have a viewpoint. Additionally, many personal rankings are not in the ranking format, *i.e.*, users use the Roundup function as a watch list. We removed rankings that do not include words, such as "Top" or "Best." We used language patterns to extract the viewpoint. For example, we created grammatical rules, such as before "–like movie" or after "movies makes me-" to extract essential parts from the ranking titles. Finally, we extracted 5,375 rankings as the training data.

We implemented the method described in section 3.2 and created a LambdaMART model that learns how a specific movie is ranked for a specific query from a feature vector of 624 dimensions. The correct labels for the features are assigned as $10, 9, 8, \cdots, 1$ in the order of their rank in individual personal movie rankings. Even if the number of movies in the ranking is less than 10, the labels are assigned in descending order starting from 10. Negative examples are assigned 0 as a label.

We used the LtR method to train the movie ranking data. We used LambdaMART [2]; as the implementation of LambdaMART, we used XGBoost Python library [4]. The parameters for learning are as follows: The objective is nDCG, eta is 0.05, gamma is 0.25, max tree depth is 12, minimal child weight is 0.5, feature-sampling rate by a tree is 0.5, alpha is 0.5, and lambda is 0.0.

4.2 Crowdsourcing Labeling

We describe the questionnaire items and crowdsourcing setting. The questions in the questionnaire ask how appropriate it is for the pair of queries and movies on a four-point scale. Examples and evaluation criteria were displayed at the top of the questionnaire page. The response criterion was 0 points if the query did not fit the movie, 1 point if it fits a little, 2 points if it fits approximately, and 3 points if the query seemed to fit overall and perfectly. Workers were allowed to search the web for a movie. Workers can enter how confident they are in their answers.

We prepared 3,200 questions, consisting of movies for the ten queries. For each query, 320 movies were obtained using TTA-D2V and the comparison method. Since the proposed method re-ranks the search results of TTA-D2V, we use the top 300 results of TTA-D2V for evaluation. Additionally, to measure how difficult the query is to retrieve, we evaluated 20 randomly selected movies.

The questionnaires for five movies for one query were combined into one crowdsourcing task. Workers answered six questionnaires, including a dummy question (a simple math problem) to prevent BOT. Each worker was allowed to answer up to 5 tasks. We collected answers from six workers for one task. Finally, we received 19,200 (3,200 tasks for six workers) answers from 1,024 workers.

Table 3: nDCG and precision in ranking sections of ranking methods (average of ten queries)

		LtR	TTA-D2V
rank part precision	1st – 30th	0.718	**0.780**
	31st – 100th	0.679	**0.737**
	101st – 300th	**0.667**	0.637
	301st – 500th	-	0.654
nDCG	@30	0.177	**0.195**
	@100	0.386	**0.425**
	@300	0.844	**0.860**

Table 4: nDCG@k for each query

Query	nDCG@30		nDCG@100		nDCG@300	
	LtR	TTA-D2V	LtR	TTA-D2V	LtR	TTA-D2V
Nostalgic	**0.212**	0.177	**0.441**	0.398	**0.870**	0.844
breathtaking Action	0.189	**0.190**	0.409	**0.431**	0.862	**0.870**
Makes me go on a trip	0.181	**0.220**	0.390	**0.456**	0.825	**0.867**
Makes me inspiring	0.174	**0.175**	0.386	**0.391**	0.878	**0.883**
Makes me hungry	0.099	**0.230**	0.274	**0.455**	0.653	**0.744**
Makes me want to guess	0.168	**0.230**	0.354	**0.470**	0.841	**0.881**
Nice music	**0.227**	0.167	**0.453**	0.377	**0.900**	0.852
Suitable for dating	0.190	**0.191**	0.411	**0.415**	**0.880**	**0.880**
Feel family love	0.172	**0.183**	0.381	**0.416**	0.868	**0.882**
Spine-chilling	0.155	**0.191**	0.362	**0.437**	0.866	**0.894**
Average	0.177	**0.195**	0.386	**0.425**	0.844	**0.860**

4.3 Result

Based on the responses obtained, we labeled each movie as to how appropriate it was for each query. Table 3 presents the average precision and nDCG of ten queries for each method. The precision is the percentage of movies that the crowd worker judges to be congruent to the query in that ranking section. In this experiment, the subjects were instructed to select 0 (different) for irrelevant movies. Therefore, the movies whose average score was one or more are considered relevant in this evaluation.

5 USER EXPERIMENT FOR SEARCH-BY-EXAMPLE

To qualitatively verify the usefulness of the example search method based on the specific information requests of searchers and user experiences, we conducted a subject experiment.

5.1 Implementation

In this experiment, we constructed an experimental website where users can search for movies by inputting their information requests in the form of movie rankings. Figure 3 shows the actual screenshots of our evaluation system. The left side of Figure 3 is a screen that allows users to input their information needs as an example ranking. Users can input up to ten movies. When the user enters the movie title partway in the text box, the movie title is suggested. Selecting

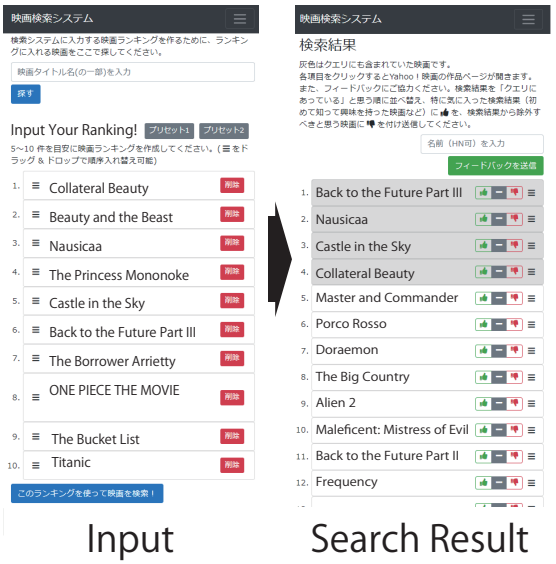

Figure 3: Screen shot of the evaluation system for search-by-example movie retrieval (Translated from Japanese)

a title will place that movie in the ranking. The user can freely reorder the ranking by drag and drop.

Pressing the search button will take the user to the results screen (right side of Figure 3). On this screen, the top 30 results of the movie search are displayed. The results will contain a mixture of user input movies and new movies. The user input movies are shaded in gray. To the right of each movie title, there is a voting button. Users can give the movie thumbs up or thumbs down if the movie is fit their search intent.

5.2 Experimental Settings

We explained to the participants the purpose of the experiment and how to use the experimental website. After that, participants can freely search for movies on the site for about one hour.

Participants could input their search intent as an example ranking. If the participant did not know a movie appeared in the search results, the participant search for the movie information on the web. At the end of the experiment, the participants were asked to answer a questionnaire.

The questionnaire asked for a ranking evaluation of whether each movie within the results was the movie they wanted and how they felt about using the system. Participants were asked to give their opinions and impressions about the accuracy and unexpectedness of the search results, the significance of expressing their requests in a ranking format, and the difference in user experience between ordinary web search and the illustrative search method.

There were five participants in this experiment. They were all male, and their age was the 20s and 30s.

5.3 Result

First, the results of the accuracy and unexpectedness of the search method are presented. We asked the participants to indicate how they felt about the accuracy of the example search method on a four-point scale, and the results are shown in Table 5.

Table 5: Answer for "How do you feel about the accuracy of the example search method?"

Choice	# participants
Very suitable	2
Suitable	1
Bad	1
Very bad	1

Two participants answered "very suitable," one "suitable," one "bad," and one "very bad." One participant, who answered "very suitable," gave the following reason: "The results were relevant not only to the genre of the movie I chose but also to the content, director, and actors." Another participant who answered "bad" gave the reason for his answer as follows: "There were not many titles that I wanted."

Next, we show the results for unexpectedness (*i.e.*, serendipity). Table 6 presents the results of a three-point test where participants were asked to indicate the number of movies, they could not discover on their own or were interested in only after being presented with the search results.

Table 6: Answer for "among the results, were there any movies that you could not discover on your own, or that you became interested in only after they were presented to you?"

Choice	# participants
Many	2
A few	3
None at all	0

Two participants answered "Many," three answered "A few," and no participant answered, "None at all." For the specific example of such search results, one participant responded as follows: "I input a movie, which is a sports story of long-distance relay road race, and 'Naoko' (a Japanese marathon movie's title) appeared in the search result. It was a movie that I had wanted to watch but had forgotten about it."

These answers suggest that although the accuracy of the search-by-example is not high, the input in the form of ranking can indicate the search intention. Additionally, if the search results contain only a few correct answers, users can discover new movies.

Next, we asked for opinions in the form of input in the movie search with the search-by-example model. We investigated whether the search intentions are conveyed and the time and effort required for input. We asked the participants about the ease of expressing information needs. For the options "keyword search is sufficient" and "example search is useful," all five answered that it was useful. There was a comment that "some search intent can only be expressed by giving examples." Then, the participants were asked

to give multiple answers to what they thought about the effort and difficulty of making queries for the search-by-example movie search. Table 7 presents the choices that one or more participants selected.

Table 7: Answer for "What was the difficult part of searching for a movie in this system?"

Choice	# participants
No problem	1
It takes a lot of time to input	2
I could not imagine movies to input	3

The most common comment from the participants was that they could not find enough movies to exemplify their search intent.

Finally, we also administered a questionnaire about the user experience of such a search system. First, we asked the participants to choose whether our search-by-example system was easier to search for unknown movies than the standard keyword search method.

Table 8: Answer for "Do you think search-by-example can find new movies more easily than a traditional keyword search?"

Choice	# participants
Strongly agree	2
Agree a little	2
Disagree a little	1
Strongly disagree	0

Most participants highly evaluated this kind of search model. One participant commented, "I felt that the ranking format allowed me to make a more personalized selection than a standard keyword search." Participants who answered "Disagree a little" to this question said, "I could not imagine movies to input, so my intention was not well communicated. Many well-known films were found."

6 DISCUSSION

Based on the results from two experiments, we discuss the keyword search and search-by-example system using LambdaMART. In the evaluation experiments of the keyword search method, the proposed method achieved lower accuracy than the existing method, Doc2Vec. Even in the top 30, the most important part of the ranking, nDCG was low at 0.177. Therefore, it is not easy to say that the results are ordered by the degree of match with the query. The proposed method was not strong enough to generate movie rankings by aggregating many people's opinions. There are two problems.

The first problem is the limitation of using LamdaMART. LamdaMART is a feature-selective machine learning algorithm based on decision trees. Therefore, it may not have understood the meaning of vectors represented by Doc2Vec in multiple dimensions correctly. To succeed in such a search task, we need to deal with information that appears across multiple dimensions. For example, vectors contain query and movie features; however, LambdaMART may not be able to consider the relationship between them. In future studies, we investigate the use of ranking learning methods based on feature generation, such as deep learning.

The second issue is the quality of the training data. In this experiment, we employed rankings created by individuals using the "Roundup" function. After the experiment, we rechecked our dataset and discovered some lists that were not ranked. Some personal rankings included the word "ranking" in their title, but their content only listed the series of films in the order of their release date. Additionally, they contained many contradictory ones since their rankings were personal. The rankings with similar titles created by different users differed in the movies they appeared in, and their rankings. Therefore, when many such personal rankings were collected and trained, the ranking model could not discover the universal features of each query well in the training data.

Next, we discuss the evaluation results of the search-by-example. From the questionnaire results, all participants could discover new movies, although there was some disagreement about the accuracy. Additionally, most participants were satisfied with the system. Even though the accuracy was low, the participants described the search results as having a "tailor-made" feel. However, many participants felt that creating rankings was laborious and complex in the search-by-example. One participant commented, "Since it takes about two hours to watch a movie, I can tolerate the time and effort required to create a query if I can avoid mistakes." However, this system is difficult to use for searchers who have not watched enough movies to create an example ranking. For example, not many users can immediately list ten movies when asked about their favorite Sci-Fi movies. Such difficulty of input is a major disadvantage of the search-by-example-based movie search.

Let us consider the reason why the satisfaction level was high despite the low accuracy. Some participants said that the results did not need to be in the form of a ranking and that they would like to see a list of more movies. Another said that it would be enough if the 30 results included only one or two movies that they wanted to see. These opinions may be because the searchers do not expect the system to have perfect order of relevancy but rather to make new discoveries. Therefore, our approach, using the LtR-based method to accept example ranking as a query, could enable a more accurate query representation of information needs.

7 CONCLUSION

This study used the LtR technique to aggregate many people's opinions and enable more accurate query input. We created a keyword search system using actual Roundup data from Yahoo! Movies and a search system based on search-by-example.

In the keyword search method, the search result candidates were sorted by the model learned from the personal rankings of many people. The evaluation experimental results showed that the proposed search method achieves lower nDCG than the simple Doc2Vec-based method, and its usefulness could not be confirmed.

In the search-by-example method, the user enters an example ranking as a search query. The system uses the ranking to train a ranking model on the fly. The results of the user experiments showed that this search method was generally well-received. The method was able to find new movies that fit the searcher's intention.

In future studies, we will improve the accuracy of the keyword search and the usability of the search-by-example model. We will use deep learning instead of LambdaMART or cleanse the training data. In the search-by-example method, we discovered that it was burdensome for users to think of ten search queries. Improvements, such as integrating the system with a keyword search or enabling search by choice are necessary.

ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI Grants Number 18K18161, 21H03775, and 18H03243.

REFERENCES

[1] O. Barkan and N. Koenigstein. 2016. Item2vec: neural item embedding for collaborative filtering. In *MLSP 2016*. IEEE, 1–6.
[2] C. J. C. Burges. 2010. From RankNet to LambdaRank to LambdaMART: An Overview. Microsoft Research Technical Report MSR-TR-2010-82.
[3] Fanglin Chen, Xiao Liu, Davide Proserpio, and Isamar Troncoso. 2020. Product2Vec: Understanding Product-Level Competition Using Representation Learning. *NYU Stern School of Business* (2020).
[4] T. Chen and C. Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proc. of SIGKDD 2016* (San Francisco, California, USA) *(KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785
[5] S. K. Karmaker Santu, P. Sondhi, and C. Zhai. 2017. On application of learning to rank for e-commerce search. In *Proc. of SIGIR 2017*. 475–484.
[6] Makoto P. Kato, Hiroaki Ohshima, Satoshi Oyama, and Katsumi Tanaka. 2009. Query by Analogical Example: Relational Search Using Web Search Engine Indices. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management* (Hong Kong, China) *(CIKM '09)*. Association for Computing Machinery, New York, NY, USA, 27–36. https://doi.org/10.1145/1645953.1645960
[7] N. Kosugi, Y. Nishihara, T. Sakata, M. Yamamuro, and K. Kushima. 2000. A practical query-by-humming system for a large music database. In *Proc. of ACMMM 2000*. 333–342.
[8] K. Kurihara, Y. Shoji, S. Fujita, and M. J. Dürst. 2019. Target-Topic Aware Doc2Vec for Short Sentence Retrieval from User Generated Content. In *Proc. of iiWAS 2019*. 463–467.
[9] Ho-Chang Lee, Hae-Chang Rim, and Do-Gil Lee. 2019. Learning to rank products based on online product reviews using a hierarchical deep neural network. *Electronic Commerce Research and Applications* 36 (2019), 100874.
[10] M. Ludewig and D. Jannach. 2019. Learning to rank hotels for search and recommendation from session-based interaction logs and meta data. In *Proc. of the RecSys Challenge 2019*. 1–5.
[11] M. Rocamora, P. Cancela, and A. Pardo. 2014. Query by humming: Automatically building the database from music recordings. *Pattern Recognition Letters* 36 (2014), 272–280.
[12] J. Salamon, J. Serra, and E. Gómez. 2013. Tonal representations for music retrieval: from version identification to query-by-humming. *IJMIR* 2, 1 (2013), 45–58.
[13] L. Wu, D. Hu, L. Hong, and H. Liu. 2018. Turning clicks into purchases: Revenue optimization for product search in e-commerce. In *Proc. of SIGIR 2018*. 365–374.
[14] J. Xu, C. Chen, G. Xu, H. Li, and E. R. T. Abib. 2010. Improving quality of training data for learning to rank using click-through data. In *Proc. of WSDM 2010*. 171–180.
[15] Yuan Yao and F. Maxwell Harper. 2018. Judging Similarity: A User-Centric Study of Related Item Recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems* (Vancouver, British Columbia, Canada) *(RecSys '18)*. Association for Computing Machinery, New York, NY, USA, 288–296. https://doi.org/10.1145/3240323.3240351
[16] Gang Yu, Junsong Yuan, and Zicheng Liu. 2013. Action Search by Example Using Randomized Visual Vocabularies. *IEEE Transactions on Image Processing* 22, 1 (2013), 377–390. https://doi.org/10.1109/TIP.2012.2216273
[17] Xinran Zhang, Xin Yuan, Yunwei Li, and Yanru Zhang. 2019. Cold-Start Representation Learning: A Recommendation Approach with Bert4Movie and Movie2Vec. In *Proceedings of the 27th ACM International Conference on Multimedia*. 2612–2616.