JYU DISSERTATIONS 100

Gaurav Pandey

Utilization of Efficient Features, Vectors and Machine Learning for Ranking Techniques

UNIVERSITY OF JYVÄSKYLÄ
FACULTY OF INFORMATION TECHNOLOGY

JYU DISSERTATIONS 100

Gaurav Pandey

Utilization of Efficient Features, Vectors and Machine Learning for Ranking Techniques

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksen Lea Pulkkisen salissa
elokuun 9. päivänä 2019 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in building Agora, Lea Pulkkinen hall, on August 9, 2019 at 12 o'clock noon.

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2019

Editors
Marja-Leena Rantalainen
Faculty of Information Technology, University of Jyväskylä
Ville Korkiakangas
Open Science Centre, University of Jyväskylä

Permanent link to this publication: http://urn.fi/URN:ISBN:978-951-39-7806-8

ISBN 978-951-39-7806-8 (PDF)
URN:ISBN:978-951-39-7806-8
ISSN 2489-9003

ABSTRACT

Pandey, Gaurav
Utilization of Efficient Features, Vectors and Machine Learning for Ranking Techniques
Jyväskylä: University of Jyväskylä, 2019, 63 p.(+included articles)
(JYU Dissertations
ISSN 2489-9003; 100)
ISBN 978-951-39-7806-8 (PDF)
Finnish summary
Diss.

Document ranking systems and recommender systems are two of the most used applications on the internet. Document ranking systems search for documents in response to a query given by the user. On the other hand, recommender systems suggest items to the users on the basis of their previously expressed preferences. Both document ranking systems and recommender systems make use of ranking techniques, since they typically present their results in the form of a ranked list. The order of the results is important because the users expect the most useful results at the top of these ranked lists.

Improvements in algorithms used by document ranking systems and recommender systems, including the utilization of advanced machine learning techniques, lead to the generation of improved rankings. Moreover, advanced document ranking systems often use features collected from the documents to generate rankings. Similarly, vectors generated for the users as well as items are utilized by the recommender systems. Therefore, generation of features and vectors of good quality is instrumental for ranking techniques.

This dissertation makes the following contributions to explore the improvements in ranking techniques using efficient features, vectors and machine learning: a) Creation of a feature extraction algorithm for learning to rank tasks in document ranking, b) Creation of pairwise preference vectors of ratings on items by using neural embeddings that can be utilized in machine learning tasks including recommender systems, c) Utilization of deep neural networks and transfer learning for serendipitous recommendations, d) Recommendations using ranking probabilities and non-negative matrix factorization and e) Application of neural embeddings to search for cities and tours, taking user's travel interests into account.

Keywords: Ranking, Information Retrieval, Recommender Systems, Deep Learning, Neural Embedding, Serendipitous Recommendations

Author

Gaurav Pandey
Faculty of Information Technology
University of Jyväskylä
Finland

Supervisors

Professor Dr. Jari Veijalainen
Faculty of Information Technology
University of Jyväskylä
Finland

Dr. Shuaiqiang Wang
Data Science Lab
JD.COM
China

Professor Dr. Mikko Siponen
Faculty of Information Technology
University of Jyväskylä
Finland

Reviewers

Associate Professor, Dr. Qipeng Phil Zheng
Department of Industrial Engineering and
Management Systems
University of Central Florida
USA

Postdoctoral Researcher, Dr. Fedelucio Narducci
Department of Computer Science
University of Bari "Aldo Moro"
Italy

Opponent

Associate Professor, Dr. Kostas Stefanidis
Faculty of Information Technology and
Communication Sciences
University of Tampere
Finland

ACKNOWLEDGEMENTS

It would not have been possible for me to complete the dissertation without the immense support that I have been fortunate to receive. Out of the many people who have supported me, in this acknowledgment, I am able to thank only a few.

First and foremost, I would like to extend my immense and sincere gratitude to my supervisors: Prof. Jari Veijalainen, Dr. Shuaiqiang Wang and Prof. Mikko Siponen for their support, guidance, encouragement and help during my doctoral studies. I would also like to thank the external reviewers of the dissertation and the opponent, for their efforts and time to review my dissertation.

Moreover, I would like to thank all the co-authors of my published articles for giving me a great opportunity to collaborate with them. Also, I would like to thank all my colleagues at the Faculty of Information Technology at the University of Jyväskylä for their support. I would like to especially thank Dr. Alexander Semenov and Dr. Denis Kotkov for their discussions and collaborations.

The organizational and infrastructural support is very important for carrying out successful research. For this, I am very grateful that over the years I have been enabled and supported at the University of Jyväskylä. I am thankful for the grants from the University of Jyväskylä, including the mobility grant as well as the Academy of Finland Grant (MineSocMed # 268078).

I would also like to thank my family and all my friends. My wife Sakshi and son Garv have been really kind to me during this research journey.

GLOSSARY

Ranking	Arrangement of a set of entities in a decreasing order of expected usefulness (e.g. relevance).
Document	Textual representation of a tangible or non-tangible entity.
Item	A piece of information referring to tangible or intangible entities on which users have given their ratings.
Document Ranking System	A software system that arranges a set of documents in decreasing order of usefulness in response to a query.
Recommender System	A software system that presents a ranking of items to a user, based on her and other users' past preferences.
Features	Information, typically real number values attached to document or query-document pairs, based on their properties.
Vectors	Ordered set of real numbers created for entities (words, items, users, etc.) using neural networks.
Neural Embedding	Use of neural networks to create vectors for words, so that semantically similar words have similar vectors. Also used to create user and item vectors.
Serendipitous Recommendations	Recommendations given to users that consist of serendipitous (i.e. relevant, novel and unexpected) items.
Vertical Search	A search functionality that specializes in a particular domain.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

LIST OF INCLUDED ARTICLES

PI Gaurav Pandey, Zhaochun Ren, Shuaiqiang Wang, Jari Veijalainen, Maarten de Rijke. Linear feature extraction for ranking. *Information Retrieval Journal*, 2018.

PII Gaurav Pandey, Shuaiqiang Wang, Zhaochun Ren, Yi Chang. Vectors of pairwise item preferences. *European Conference on Information Retrieval*, 2019.

PIII Gaurav Pandey, Denis Kotkov, Alexander Semenov. Recommending serendipitous items using transfer learning. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018.

PIV Gaurav Pandey, Shuaiqiang Wang. Listwise recommendation approach with non-negative matrix factorization. *Intelligent Decision Technologies 2018*, 2018.

PV Mohamed Abdel Maksoud, Gaurav Pandey (first co-author), Shuaiqiang Wang. CitySearcher: A city search engine for interests. *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017.

PVI Mohamed Abdel Maksoud, Gaurav Pandey (first co-author), Shuaiqiang Wang. Finding tours for a set of interests. *Companion Proceedings of The Web Conference 2018*, 2018.

1 INTRODUCTION

With the ongoing increase in internet usage as well as the growth in the content available online, various technologies are being continuously developed to assist the users in accessing this large quantity of content. Document ranking systems and recommender systems are such technologies and two of the most widely used web applications nowadays. A user could present a query to a search engine, and the *document ranking system* would get the user a list of webpages that satisfy her needs. Moreover, users get automatic recommendations on various websites using *recommender systems*, based on their previous preferences and actions. For example, on an e-commerce website the automatic recommender system would suggest the user with products that she might be interested in. Both document ranking systems and recommender systems typically present their results in the form of a ranking. The order of the results is important for both of them, because the user expects that the most useful results would occur at the top of the ranking.

While the field of information retrieval consists of various domains such as text clustering, summarization, topic detection, etc., document ranking is its prime task (Büttcher et al., 2016). Document ranking (or document search) aims to present a ranked list of documents from an available document collection, in response to a query posed by the user. In a web search system, the webpages available on the internet typically form the document collection. Google [1], Bing [2] and Yahoo! Search [3] are some of the widely used search engines that focus on webpage rankings. Apart from web search, document ranking is often used in various other domains such as product search on an e-commerce website or book search in a digital library. The search on e-commerce websites is performed on documents that could be representative of both tangible (physically existing) as well as non-tangible (e.g. multimedia, services, etc.) products. In general, document ranking system could be used on any given type of document collection. Some of the popular e-commerce sites where a ranking of the products is gener-

[1] https://www.google.com/
[2] https://www.bing.com/
[3] https://search.yahoo.com/

ated in response to a query from the user are: Amazon [4], eBay [5], Alibaba [6] and Zalando [7].

In the context of this dissertation, we define a *document ranking system* as a software system that generates a ranking of the documents in response to a *query* given by the user. Here, the document refers to a textual representation of an entity, where the entity can either be a tangible good or digital object. The textual representations of various entities are created or formatted so that they can be utilized by software such as document ranking systems. For example, tangible goods could be various products such as electronics, clothes, etc., that have a textual counterpart on e-commerce websites. The information such as product description, name, price, producer, etc., in the form of text would form the document for a document ranking system in product search domain. Similarly, for digital objects like general webpages, news articles, domain specific texts, etc., the text present in them would form the documents for the document ranking system in web search domain. The textual descriptions for non-tangible goods such as courses, e-books, services, movies, etc., can also be considered as documents. Indeed, non-textual information like videos, images, audio, etc., could also be utilized by document ranking systems; but in our work we restrict the scope only to textual content in documents. In addition, features can also be collected for the document, that can be used by the document ranking system. For example, in web search some of the commonly used features are: document length, title length, PageRank, etc. Moreover, features could be for a document as well as for a query-document pair.

Recommender systems on the other hand, present the user with a personalized list of items as recommendations, based on the past user behavior, as far as it is known to the recommender system through the interactions with the user (Ekstrand et al., 2011). Unlike document ranking systems, there is no explicit query needed from the user. To provide recommendations to a particular user, the recommender system utilizes the past behavior of the user along with the behaviors of other users. The recommendations are provided automatically to the users, for example as an additional functionality on the webpage or as an email sent to them. The past behavior that is utilized, is some indicator of users' liking for the items; e.g. ratings, views, clicks, likes, etc. The recommendations typically consist of a ranking of items that the user has not accessed before. For instance, a movie recommender system would use the users' ratings of the movies along with the ratings of the other users, to recommend her a list of movies out of the movies that she has not rated before, and thus, has probably not seen before. Recommender systems can also use information about the user (e.g. age, occupation, expressed genre preferences, etc.) as well as the movies (e.g. genres, year of release, language, etc.) to make the recommendation. Most of the popular e-commerce websites such as Amazon, eBay, Alibaba etc., provide personalized

[4] https://www.amazon.com/
[5] https://www.ebay.com/
[6] https://www.alibaba.com/
[7] https://www.zalando.com/

recommendations for the users, based on the purchase or rating history of the user and other users. Some of the popular websites sites that provide movie recommendations are: Netflix [8], MovieLens [9], IMDb [10], etc.

Specifically in context of this dissertation, we define a recommender system as a software system that generates a ranking of items to be recommended to the user, by utilizing the past ratings given by the users on items. Here, item means a piece of information referring to tangible or intangible entities on which the users have provided their ratings. Depending on the context, the term item is also used in our discussions to refer to the entity itself. The available ratings are the main information about the item, that is considered by us for the purpose of giving recommendations to the users. For example, in a movie recommendation system, the movies are the items on which the users provide their ratings (e.g. a highest rating of 5 for a liked movie and the lowest rating of 1 for a movie that is disliked). Similarly, in case of product recommendation, the items could be tangible products such as furniture, clothes, etc., or non-tangible products such as e-books, healthcare service, online course, etc. Apart from ratings, recommender systems can also utilize user preferences in other forms, e.g. clicks on item links, likes, comments etc., as well as textual descriptions and meta-data related to the users and the items. However, this dissertation focuses mainly on the utilization of the explicit ratings given by the users in the past for recommendations. The other information about the users and items (e.g. textual descriptions, metadata, etc.) is also not considered for recommendations in this work.

It should be noted that we use the term *document* in relation to document ranking systems, while the term *item* is used in the context of recommender systems; where documents and items could refer to any entity that is been searched or recommended, respectively. In fact, they could also refer to the same entity. For example, on an e-commerce website like Amazon, the products can be searched (i.e. treated as documents by document ranking system) and are also recommended (i.e. treated as items by recommender system). However, in the context of this dissertation, the key difference between document and item is the information regarding the entity that is utilized along with the system that is using the information. Specifically, the textual information from the documents is utilized by the document ranking system. And, the past user ratings on the items are utilized by the recommender systems. Of course, it is possible for document ranking systems to utilize user behavior on documents (e.g. ratings, clicks, etc.) and recommender systems to utilize the textual descriptions related to items; but these topics are not addressed in the dissertation.

As mentioned, recommender systems as well as document ranking systems present the user with a ranking of items or documents. To create a ranking of the results that satisfy the user needs, is the main aim of ranking techniques used in recommender systems (Adomavicius and Tuzhilin, 2005) and information retrieval systems (Liu, 2011; Joachims et al., 2007). The order of the items or docu-

[8] https://www.netflix.com/
[9] https://movielens.org/
[10] https://www.imdb.com/

ments in the ranking is important because the user's attention goes more to the items at the top of the ranking. Therefore, it is desirable that the item or document that fulfills the user's needs the most is shown first, followed by the others in a decreasing order of user satisfaction.

The ranking of documents returned by a document ranking system should show the most useful document at the top followed by the second most useful document and so on. For example, if a user searches for the query "Finnish cities", then she would expect the top results to be highly relevant to her search needs. She would expect to see some documents that are related to the topic of cities in Finland and are able to impart information about them. If the top results are useless for her, then one could expect that she would not be satisfied with the document ranking system. The widely used search engine Google, provides the search results in the form of a ranking.

Similarly, a recommendation system should rank the most useful item to the user at the top, followed by other items in decreasing order of usefulness. For instance, in a movie recommendation system, the movies in the beginning of the list of the recommended movies should be the most useful to the user, and hence have a good chance to be watched. Similarly, in a recommender system on an e-commerce website, the list of recommended items should be created aiming that the user would find the items in the beginning of the list the most useful, and hence would like to buy them.

Since in a ranked list of documents / items, it is expected that the ranking is in decreasing order of usefulness to the user, where the first one is the most useful; it is typically one of the main requirements of document ranking systems and recommender systems to have as good ranking performance as possible. The aforementioned *usefulness* of the ranked items or documents to the users can vary according to the functionality aimed by the recommender system or document ranking system. Often the usefulness is considered to be the *relevance* of the presented documents with respect to user query in document ranking systems and the *relevance* of the presented items to the user in recommender systems. However, the user needs could be specific to a particular domain. Also, the ranking systems may aim to provide serendipitous results to the users (André et al., 2009; Kotkov et al., 2016).

Moreover, advances in machine learning have led to a widespread use of neural embedding techniques to create *vectors*. These vectors can be considered to be an ordered set of real numbers, which represent an entity. Vectors for words in documents (Mikolov et al., 2013b,a) can be created by neural embedding by using the textual content of documents, and then used for document ranking. Also, vectors for users and items can be created by using the ratings and then these vectors can be utilized by recommender systems (Barkan and Koenigstein, 2016; Grbovic et al., 2015). While the features that are collected from the documents can also be an ordered set of real numbers like in the case of vectors, we specifically use the term vector when such representation for an entity is created using neural embedding.

1.1 Motivation

This section presents the motivation for the research work presented in this dissertation.

Various learning to rank techniques (Cao et al., 2007; Freund et al., 2003; Joachims et al., 2009; Niu et al., 2012; Xu and Li, 2007), that utilize machine learning, are used for document ranking. They typically use machine learning to train a ranking model on an available training dataset, that contains a set of queries and a list of documents for each query, where relevance levels are available for each query-document pair. For example, a training set with binary relevance levels would have levels 0 and 1 assigned to query-document pairs, corresponding to relevant and non-relevant instances, respectively. The relevance levels can also be in higher order of granularity. Moreover, features are defined and created, that are typically a set of real number values corresponding to each document or query-document pair. They are generally created in order to be utilized by machine learning algorithms and hence are available to the learning to rank algorithms. Now ranking models can be trained using these features, and the trained model can thereafter be used to predict the relevance of unseen documents and therefore rank them (Joachims et al., 2007).

These learning to rank algorithms aim to utilize more and more useful features of the documents in order to improve the ranking (Liu, 2011). In the presence of high number of features, dimensionality reduction of the available document features becomes a crucial task, where original features are used to form a lesser number of new features. This is because, when a smaller set of more discriminative and less redundant features are selected or generated for learning, the effect of overfitting is reduced leading to higher accuracy (Ng, 2004). Moreover, since dimension reduction leads to a lesser number of features, this makes training and prediction process more computationally efficient (Liu, 2011). Dimension reduction can be achieved by feature *selection* and feature *extraction*. Feature selection selects a subset of the original features for learning, whereas feature extraction uses the original features to generate a smaller set of new features. There are various efforts to investigate feature selection for ranking (Geng et al., 2007; Gupta and Rosso, 2012; Lai et al., 2013; Laporte et al., 2014; Naini and Altingövde, 2014; Pan et al., 2009; Yu et al., 2009). However, to the best of our knowledge, there are no investigations in feature *extraction* for learning to rank. Therefore, we aim to present an algorithm to carry out feature extraction for learning to rank.

Secondly, neural embedding has evolved as an advanced machine learning technique that creates vectors of words from document collections and is being utilized in many applications (Socher et al., 2011; Turney, 2013; Collobert et al., 2011; Turian et al., 2010; Kusner et al., 2015). The creation of vectors by neural embedding is based on the presumption that the words that occur close to each other in the documents are more closely related than the words that occur far away. *word2vec* vectorization algorithm (Mikolov et al., 2013a,b) has recently gained immense attention because of its efficiency. It has also been utilized in

the area for document ranking (Nalisnick et al., 2016; Ye et al., 2016; Ganguly et al., 2015). There is scope to explore niche document ranking domains of vertical search, i.e. focusing on specific type of document collection. One such domain that can be explored is vertical search for travel interests. In such vertical search, travel destination documents representing one city per document are searched for queries regarding travel interests, by utilizing vector representations of the words present in the documents. Therefore, the main purpose is to create a ranking of cities or tours (group of cities) in response to travel interest queries.

Moreover, *word2vec* has also also been successfully applied to recommendation scenarios like: *prod2vec* (Grbovic et al., 2015) and *item2vec* (Barkan and Koenigstein, 2016). In these algorithms, in order to create vectors of the items, each user is considered as a document, and each item on which the user has performed a particular action (e.g. rating, purchase, click, like, etc.) is considered as a word contained in the document. This enables these algorithms to create vectors of items in a way similar to the creation of vectors of words by *word2vec*. However, in the input each item can only have two possible states for a user: 1 or 0, representing action taken by the user or missing action, respectively. Though these representations are expected to create good quality item vectors for some tasks involving binary actions by the users (e.g. views, likes, etc.), they lack the functionality to capture higher levels of granularities of users' feedback, e.g. ratings on a scale of 1 to 5. This is because, such binary representations would treat a low rated item in the same way as a highly rated item. Hence, it is expected to severely limit the vectorization quality. Therefore, there is the need to investigate the neural item embedding problem, to create good quality vectorization for items using users' historical rating information while utilizing the higher granularities of user feedback.

Thirdly, rating-oriented approaches are followed by conventional recommendation algorithms. In order to learn a recommendation model, they typically consider users' observed historical ratings, to predict their ratings for items with unknown ratings. Many studies state that the prediction of accurate ranking is of higher importance than the prediction of accurate rating scores (Gunawardana and Shani, 2009; Adomavicius and Tuzhilin, 2005). In addition, accurate rating prediction does not necessarily lead to accurate ranking results in recommender systems. It is possible that a recommender system achieving high accuracy in predicting the ratings does not predict correct rankings. Matrix factorization (Koren et al., 2009) is one of the most fundamental and widely used technique used in recommender systems. Although, there are various ranking-oriented recommendation algorithms, most of the pioneering efforts on *ranking-oriented* matrix factorization predict users' item ranking based on rating scores. Therefore, there is need to explicitly present users' preference ranking on items for matrix factorization.

Fourthly, serendipitous recommendations are gaining much attention these days. The need for serendipity arises because relevance oriented recommender algorithms suggest items that the user already knows about or could find easily herself (Kotkov et al., 2016). This may lead to low user satisfaction and formation

of a filter bubble (Nguyen et al., 2014). Hence, recommendation of serendipitous i.e. a combination of relevant, novel and unexpected items, can be expected to broaden the user's preferences (Kotkov et al., 2018). While deep learning and advanced machine learning algorithms have been used in various algorithms for relevance oriented recommendations (He et al., 2017; Grbovic et al., 2015; Barkan and Koenigstein, 2016), the efforts are very limited for serendipitous recommendations. Therefore, there is need to explore utilization of deep learning algorithms for serendipitous recommendations. Moreover, there are no publicly available large serendipity oriented datasets, to the best of our knowledge. Therefore, there is also the need to explore transfer learning, so that a deep neural network can be pre-trained on a different large dataset (e.g. relevance oriented dataset) and then can be tuned for a smaller serendipity oriented dataset.

1.2 Research Questions

Based on the presented motivations, this dissertation primarily tries to address the following research questions. Along with the research questions, we also briefly introduce the way they have been addressed by the research presented in this dissertation.

RQ1: In document ranking, how can good quality features be extracted for learning to rank?

This research question has been addressed in Article PI where we focus on the feature extraction problem for learning to rank for document ranking. For this, we propose *LifeRank*, a linear feature extraction algorithm that utilizes original features to generate new features for documents, while aiming to match the learning to rank problem. Our experiments on benchmark datasets show improvements over the state-of-the-art techniques.

RQ2: How can neural embedding be used to create good quality item vectors, while utilizing different levels of granularities of user ratings?

This research question has been addressed in Article PII, where we propose a vectorization algorithm called *Pref2Vec* that is capable of utilizing different levels of granularities in given user ratings. For this, the algorithm creates the vectors for pairwise item preferences, that can be utilized to create good quality item vectors as well as user vectors. Experiments on benchmark datasets confirm the quality of the created vectors.

RQ3: How can deep learning and transfer learning be utilized for serendipitous recommendation?

This research question has been addressed in Article PIII, where we present our novel transfer learning algorithm *SerRec* to recommend serendipitous items to the users. The algorithm first trains a deep neural network using

a large dataset with relevance scores and then tunes the trained network using a smaller dataset with serendipity scores. The algorithm shows improvements over the state-of-the-art serendipity oriented algorithms.

RQ4: How can users' preferences be formulated in listwise fashion and utilized for ranking oriented recommendations?

This research question has been addressed in Article PIV. The article formulates a problem that aims to predict the probabilities of item rankings for the users, by utilizing a user-ranking probability matrix. The recommendations generated using this algorithm on benchmark datasets, demonstrate the effectiveness of the algorithm.

RQ5: How can we utilize neural embeddings of words for innovative ranking applications in vertical search?

This research question has been addressed in Articles PV and PVI. Although there could be various ways in which neural embeddings of words could be used, we explored the relatively unexplored vertical search domain of finding cities and tours for travel interests of a user. In Article PV, we present the algorithm that utilizes *word2vec* to rank cities for a travel interest. Moreover, it uses a technique to reduce the effect of mismatched semantic results in the rankings, by generating features for the documents using a novel clustering algorithm. The algorithm shows improvements over standard techniques. Furthermore, Article PVI utilizes this algorithm of generating rankings of cities for interests, in order to generate a ranking of tours (i.e. groups of nearby cities) for a set of travel interests. This is demonstrated as a web application.

1.3 Research Methods and Process

For addressing the research questions and to finally create the corresponding research articles, in each case the research methods were followed in similar fashion. For a particular article, literature review was carried out in order to know the state-of-the-art and to identify the research gap. Thereafter, constructive method was followed to create new algorithms, along with their software implementations. Then by running the algorithms on benchmark databases, offline evaluation was carried out, that included comparisons with the state-of-the-art algorithms.

The research presented in this dissertation has followed the Design Science approach (Peffers et al., 2007), that consists of the following six stages: (1) problem identification and motivation, (2) definition of the objectives for a solution, (3) design and development, (4) demonstration, (5) evaluation and (6) communication. We now present the way research was carried out in our six articles in accordance to these stages. While research, corresponding to multiple articles,

was sometimes carried out in parallel, for each article the six stages occurred sequentially.

Article PI "Linear feature extraction for ranking"

Problem identification and motivation: There are no known efforts in extracting document features from known features, for learning to rank. Such feature extraction can improve the accuracy as well as efficiency of learning to rank algorithms.

Definition of the objectives for a solution: The objective is to create a feature extraction algorithm that is suitable for learning to rank tasks.

Design and development: The *LifeRank* algorithm was designed and developed.

Demonstration: *LifeRank* algorithm was used on benchmark datasets for feature extraction and the extracted features were then used by standard learning to rank algorithms.

Evaluation: The improvements in performance over standard feature selection algorithms were shown using offline evaluations.

Communication: Publication in Article PI is the primary communication.

Article PII "Vectors of pairwise item preferences"

Problem identification and motivation: Existing neural embedding techniques for creating item vectors do not consider the different levels of granularities in the ratings given by the users on items. Incorporation of the different granularities is expected to create higher quality item vectors.

Definition of the objectives for a solution: The objective is to create a neural embedding algorithm that enables utilization of granularities of ratings on items, in order to create good quality item vectors.

Design and development: The *Pref2Vec* algorithm was designed and developed.

Demonstration: Using standard benchmark datasets, the *Pref2Vec* algorithm was used to create vectors of pairwise item preferences, that in turn were used to create user vectors and item vectors. The algorithm to generate recommendations to users using these vectors was also demonstrated.

Evaluation: The improvements in quality over standard vectorization techniques were shown using offline evaluations.

Communication: Publication in Article PII is the primary communication.

Article PIII "Recommending serendipitous items using transfer learning"

Problem identification and motivation: There are no existing techniques that use deep learning and transfer learning to generate serendipitous recommendations using serendipity oriented datasets. Such a technique is expected to improve the performance in generating serendipitous results.

Definition of the objectives for a solution: The objective is to fill the identified research gap by creating an algorithm that uses deep learning and transfer learning to generate serendipitous recommendations.

Design and development: The *SerRec* algorithm was designed and developed.

Demonstration: Using two benchmark datasets with relevance scores and serendipity scores, *SerRec* was demonstrated to generate serendipitous recommendations using transfer learning.

Evaluation: Improvements were shown over standard serendipity oriented recommendation techniques using offline evaluations.

Communication: Publication in Article PIII is the primary communication.

Article PIV "Listwise recommendation approach with non-negative matrix factorization"

Problem identification and motivation: Recommender systems aimed at high accuracy in predicting the ratings do not necessarily predict correct rankings. Most ranking-oriented recommendation algorithms that use matrix factorization predict users' item ranking based on rating scores. Therefore, it is needed to formulate users' preference ranking on items and utilize it for matrix factorization.

Definition of the objectives for a solution: The objective is to create an algorithm that utilizes and predicts ranking probabilities of items in matrix factorization and then use them to make recommendations.

Design and development: The *LwRec* algorithm was designed and developed.

Demonstration: Using standard benchmark datasets, the *LwRec* algorithm was used to generate recommendations for users.

Evaluation: Improvements were shown over standard recommendation techniques using offline evaluations.

Communication: Publication in Article PIV is the primary communication.

Article PV "CitySearcher: A city search engine for interests"

Problem identification and motivation: While using *word2vec* to find ranking of cities for travel interests, we have the problem of mismatched semantic relationships, i.e. travel interests get matched to certain words in the documents incorrectly.

Definition of the objectives for a solution: The objective is to utilize *word2vec* for finding ranking of cities for a travel interest while reducing the effect of mismatched semantic relationships.

Design and development: The *CitySearcher* algorithm was designed and developed.

Demonstration: The *CitySearcher* algorithm was used on a publicly available travel dataset containing documents regarding cities.

Evaluation: Improvements were shown over standard document ranking techniques using offline evaluations.

Communication: Publication in Article PV is the primary communication.

Article PVI "Finding tours for a set of interests"

Problem identification and motivation: There are no exiting algorithms for creating a ranking of tours for a set of interests by using available ranking of cities for interest (from Article PV) while aiming that each interest from the set of interests is satisfied. Here each tour is defined as a set of cities.

Definition of the objectives for a solution: The objective is to create an algorithm, to use ranking of cities for an interest, in order to form ranking of tours for a set of interests.

Design and development: The tour generation algorithm was designed and developed.

Demonstration: The developed algorithm was used on a travel dataset.

Evaluation: Comparison was done on the variants of the proposed algorithm using offline evaluations.

Communication: Publication in Article PVI is the primary communication. Moreover the web-application is presented on Travición website [11].

Table 1 summarises our research questions, along with the research methods used to address them.

[11] https://www.travicion.com/

TABLE 1 Summary of Research Questions and Research Methods

Research Question	Problem	Solution Objective	Development	Demonstration	Evaluation	Communication
RQ1: In document ranking, how can good quality features be extracted for learning to rank?	No known efforts to extract document features for learning to rank	Algorithm required	*LifeRank*	Used on benchmark dataset to extract low dimension features that were used by standard learning to rank methods	Offline Evaluation: Comparison with standard feature selection techniques	Article PI
RQ2: How can neural embedding be used to create good quality item vectors, while utilizing different levels of granularities of user ratings?	Existing neural embedding techniques do not consider levels of granularities in ratings	Algorithm required	*Pref2Vec*	Used on benchmark dataset to create preference vectors, that were used to create user vectors and item vectors	Offline Evaluation: Comparison with standard vectorization techniques	Article PII
RQ3: How can deep learning and transfer learning be utilized for serendipitous recommendation?	No existing techniques to use deep learning for serendipitous recommendations	Algorithm required	*SerRec*	Used on benchmark dataset, to train deep neural network using relevance scores and then tune for serendipity scores	Offline Evaluation: Comparison with standard serendipity oriented algorithms	Article PIII
RQ4: How can users' preferences be formulated in listwise fashion and utilized for ranking oriented recommendations?	No existing technique to formulate users' preference ranking on items and utilize it for matrix factorization	Algorithm required	*LwRec*	Used on benchmark datasets to generate recommendations	Offline Evaluation: Comparison with standard recommendation algorithms	Article PIV
RQ5: How can we utilize neural embeddings of words for innovative ranking applications in vertical search?	Using *word2vec* to find city rankings for travel interests leads to mismatched semantic relationships	Algorithm required	*City-Searcher*	Used on travel dataset to search for cities	Offline Evaluation: Comparison with standard retrieval techniques	Article PV
	Need to create ranking of tours for a set of interests by using available ranking of cities for interest	Algorithm required	Tour Creation	Used on travel dataset to search for tours	Offline Evaluation: Comparison among variants of proposed algorithm	Article PVI

1.4 Thesis Structure

The dissertation is structured as follows. Chapter 2 discusses the related work and then the results are described in Chapter 3. Chapter 4 summarises the original articles that are included in this dissertation. Then Chapter 5 concludes the dissertation, and is followed by the original articles.

This dissertation includes six original articles. All of them are published, at the time of writing the final dissertation.

2 RELATED WORK

In this chapter, we present the literature review regarding the research carried out for this dissertation.

2.1 Learning to Rank for Document Ranking

This section describes the related work in the area of learning to rank and feature selection.

2.1.1 Learning to Rank Methods

Learning to rank methods use machine learning algorithms and find application in document ranking. Due to this, machine learning as well as information retrieval communities are becoming increasingly interested in learning to rank. Most of the efforts in the area of learning to rank have focused on offline learning to rank. Here, explicit feedback given by the users in the past for query-document pairs, mostly labels in the form of relevance scores, is used for training a learning to rank model. Because of the effectiveness of such models, a number of algorithms have been developed. The offline learning to rank algorithms can be divided into three main classes (Liu, 2009; Chapelle et al., 2011): pointwise, pairwise, and listwise.

Pointwise approaches consider each labeled document as a training instance, for a particular query. Here each training instance is considered as independent from other training instances. For the query, the approach learns a model by using a machine learning technique like classification or regression. The model can then be utilized to predict the relevance scores of unlabeled documents. Now, using these predicted scores, the unlabeled documents can be ranked for the query. Pointwise approaches are used by various algorithms such as Pranking (Crammer and Singer, 2001), McRank (Li et al., 2007) and Subset Ranking (Cossock and Zhang, 2008).

Pairwise approaches consider each pair of documents as a training instance. Using this, the approach learns a binary classifier, that can then be used to predict preference for a pair of unlabeled documents, signifying which document out of the two would have higher relevance score. These predicted preferences can then be aggregated to form a ranking of the unlabeled documents. Some of the well-known pairwise ranking algorithms are: Ranking SVM (Joachims et al., 2009; Cao et al., 2006), RankBoost (Freund et al., 2003), RankNet (Burges et al., 2005), FRank (Tsai et al., 2007), LambdaRank (Burges et al., 2007), and BoltzRank (Volkovs and Zemel, 2009).

Listwise approaches consider the ranked list of labeled documents as a training instance. Using this, they try to learn a model that can then be used to predict the ranking of all the documents for the query. Some of algorithms that use the listwise approach are: ListNet (Cao et al., 2007), SVM-MAP (Yue et al., 2007), NDCGBoost (Valizadegan et al., 2009), etc.

Apart from offline learning to rank, *online* learning to rank (Schuth et al., 2016; Hofmann et al., 2013; Grotov and de Rijke, 2016; Zoghi et al., 2017) and *counterfactual* learning to rank from online data (Joachims et al., 2018) have been gaining attention recently. These topics however are not considered in this dissertation.

2.1.2 Feature Selection Methods for Learning to Rank

The learning to rank methods for document ranking utilize the features provided for the documents along with their relevance scores with respect to a query, in order to learn a model. These features are typically real number values that correspond to the properties of the document or query-document pair. The aim to improve the ranking performance of such models, leads to the inclusion of more and more useful features. Because of potentially very large number of features, dimension reduction, i.e. reducing the number of features of the documents, has emerged as an important issue in the ranking problem (Geng et al., 2007). There are various benefits that could be achieved because of dimensionality reduction. Firstly, because of fewer features the learning to rank algorithms would run faster, leading to higher efficiency. Secondly, the accuracy of the learning to rank algorithms could also be increased because an effective dimensionality reduction algorithm would lead to discriminative features with less redundancy and noise. Moreover, using such features in learning would reduce the chances of overfitting in the learnt model (Ng, 2004).

Feature selection and feature extraction are the two ways in which the dimensionality of the document features can be reduced. In feature selection, for the given original features, a subset of features is chosen. But in feature extraction, using the original features, a smaller number of entirely new features are created. For example, a new feature could be a linear combination of some of the original features. Various efforts have been carried out recently for feature selection for learning to rank. GAS (Geng et al., 2007) is one of the initial algorithms for this, that aims to incorporate the importance and similarity of features

in order to achieve feature selection for ranking. Other prominent efforts for feature selection have been carried out by Metzler (2007), Pan et al. (2009), Yu et al. (2009), Gupta and Rosso (2012), Lai et al. (2013), Naini and Altingövde (2014) and Laporte et al. (2014).

We see that all these algorithms aim at *feature selection* for ranking. This leads to our RQ1, since to the best of our knowledge, there are no efforts aim at *feature extraction* for ranking. We have addressed this research question in Article PI (Pandey et al., 2018b).

2.2 Recommender Systems

Various recommender systems have been introduced that aim to present the users with a ranking of items that the user has not consumed previously. The recommender systems fall typically into one of the three categories: collaborative filtering, content-based filtering and hybrid.

Collaborative filtering utilizes the ratings given by the users in the past on items to make recommendations. They do not need other information related to the items or the users. Content-based filtering algorithms utilize the descriptions for the items in order to select items to be recommended to the user (Pazzani and Billsus, 2007). They aim to match up the attributes from the existing user profile having the user's preferences, with the attributes of an item, to generate recommendations (Lops et al., 2011). Hybrid recommendations combine the collaborative filtering and content-based filtering approaches (De Campos et al., 2010).

The research presented in this dissertation focuses only on collaborative filtering. Based on whether the collaborative filtering recommender system algorithms aim to predict the ratings or the rankings, they can be divided into two main categories: *rating oriented* and *ranking oriented*, respectively.

2.2.1 Rating Oriented Algorithms

Rating oriented collaborative filtering algorithms aim to predict accurate ratings of the unrated items for the user, that can later be used to generate a ranking of recommended items. Moreover we further classify these algorithms into *memory-based* and *model-based*. Memory-based approaches make predictions on the direct utilization of ratings. Whereas, model-based approaches use an algorithm to learn a model in the training process by utilizing available ratings. Then this learnt model is used to make predictions for unrated items.

Memory-based rating oriented algorithms are either user-based collaborative filtering or item-based collaborative filtering. User-based approaches (Herlocker et al., 2002) use the similarities between users on the bases of ratings given by them. Whereas, item-based approaches (Linden et al., 2003), utilize similarities between items on the bases of ratings given to them. Examples of advanced

implementations of item-based rating oriented approaches are: SLIM (Ning and Karypis, 2011) and FISM (Kabbur et al., 2013).

The traditional form of model-based rating oriented algorithm is matrix factorization (Koren et al., 2009). The user ratings for items are typically available in the form of a rating matrix, where each rows represents a user and a column represents an item. The rating matrix generally has various values missing. Matrix factorization uses two lower dimensional matrices whose product forms the predicted rating matrix; and the algorithm aims to decrease the distance between predicted and observed rating matrices. Probabilistic MF (Salakhutdinov and Mnih, 2007), Non-negative MF (Lee and Seung, 2000), Factorization Machines (Rendle, 2010, 2012), Hierarchical Poisson MF (Gopalan et al., 2015) and LLORMA (Lee et al., 2013) are some of the recommendation techniques based on matrix factorization.

2.2.2 Ranking Oriented Algorithms

Ranking oriented collaborative filtering algorithms aim to predict the accurate ranking of items for the user, and do not go through the intermediate step of predicting the ratings of the items. These can be further classified as pairwise and listwise.

The pairwise approaches predict the pairwise preferences of items for the user and then aggregate these preferences to form a ranking. Two algorithms that are based on this approach are: EigenRank (Liu and Yang, 2008) and VSRank (Wang et al., 2014). Although pairwise approaches show better recommendation performances than the rating oriented approaches, they are less efficient because generally the number of preference pairs is much larger than the number of ratings for a user. Moreover, listwise approaches aim to optimize a ranking oriented objective function to train a model, that can then be used to predict a ranking of unrated items as recommendations. Some of the algorithms that use a similar approach are: CLiMF (Shi et al., 2012), CoFiRank (Weimer et al., 2007), BPR (Rendle et al., 2009) and GBPR (Pan and Chen, 2013).

Most ranking-oriented recommendation algorithms based on matrix factorization use rating scores in order to predict the ranking of items. We identify the research gap, that there is a need to formulate users' preference ranking on items and utilize it for matrix factorization. This leads to the formation of RQ4 that is addressed in Article PIV (Pandey and Wang, 2018).

2.3 Neural Embedding

In this section, we describe the existing work related to neural embeddings of words along with their adaptations in the area of recommender systems.

2.3.1 *word2vec*

Most of the machine learning tasks require good quality vectors related to the data they are working on, in order to achieve high performance. That is why vectorization techniques are of importance to them, since they generate vectors having discriminative and independent components. Neural embedding techniques are the vectorization techniques that are based on neural networks. In the area of natural language processing, they are used to create vector representation of words present in a document collection, and have found various applications (Socher et al., 2011; Turney, 2013; Bengio et al., 2003; Collobert et al., 2011; Turian et al., 2010; Zou et al., 2013; Schwenk, 2007). The neural embedding techniques create the vector representations of words by assuming that the words that occur in close proximity to each other in the documents are more closely related, in comparison to the words that do not occur in close proximity. The main problem was that the traditional neural embedding lacked the efficiency in training the model.

However, the widely used *word2vec* technique that was introduced a few years ago, has made the neural embedding process very efficient (Mikolov et al., 2013b,a). This is because it employs a skip-gram language model, that does not need dense matrix multiplications like the previously existing methods (Mikolov et al., 2013b). Apart from being fast to train, the skip-gram language model is highly scalable and preserves the semantic relationships of the words in their vector representations. For example, for the created word vectors it was shown that *vec("Madrid") - vec("Spain") + vec("France")* has *vec("Paris")* as its closest vector (Mikolov et al., 2013b), where *vec(x)* represents the vector corresponding to the word x. Similarly, it was found that closest vector to *vec("King") - vec("Man") + vec("Woman")* was *vec("Queen")* (Mikolov et al., 2013a).

This technique has been utilized in various applications like name entity resolution (Lample et al., 2016), word sense detection (Bhingardive et al., 2015), vector representation of biological sequences (Asgari and Mofrad, 2015; Ng, 2017), etc. Moreover, the *word2vec* approach has been extended in *paragraph2vec*, to create hierarchical neural embedding frameworks, by using vectorization of paragraphs along with vectorization of words contained in each paragraph (Djuric et al., 2015).

Vector representations of variable length texts like sentences and documents have been created by Le and Mikolov (2014). Moreover, efforts have been made to embed entities and relationships of multi-relational data in low-dimensional vector spaces (Bordes et al., 2013; Socher et al., 2013). These embeddings can then be utilized for tasks like text classification and sentiment analysis. There are also recent efforts to create neural embeddings corresponding to nodes in graphs (Perozzi et al., 2014; Grover and Leskovec, 2016).

The *word2vec* technique has been utilized in the area for document ranking (Nalisnick et al., 2016; Ye et al., 2016; Ganguly et al., 2015), where it has shown improvements in ranking performances. There is scope to utilize the technique in various types of document ranking applications. Inspired by this, we formed

RQ5, where we aim to show the utilization of *word2vec* for vertical search for cities and tours in response to user's travel interests. This research question is addressed in Article PV (Abdel Maksoud et al., 2017) and PVI (Abdel Maksoud et al., 2018).

2.3.2 Neural Embedding for Recommender Systems

The success of *word2vec* in creating vector representations of words has also inspired its applications in the area of recommender systems. For example, Grbovic et al. (2015) presented *prod2vec* and *user2vec*. Here, *prod2vec* model utilizes neural embedding on sequences of purchased products (same as items) by the users, considering each product as the counterpart of word in the natural language processing domain, to create vectors of products. The *user2vec* model is inspired by Le and Mikolov (2014) and considers a user as a global context and hence learns the vector representations of user and products (Grbovic et al., 2015). *item2vec* (Barkan and Koenigstein, 2016) is another similar approach that considers sets of items on which the user has taken action (e.g. products purchased) and ignores the sequence of the considered items. Then it employs neural embedding of these sets of items, in order to create the item vectors. These techniques lead to creation of good quality vectors that can be used in various machine learning tasks, including recommender systems.

However, we see that all these existing techniques use item information in the binary format of 0 and 1 (e.g. clicked and not clicked song, purchased and not purchased items, etc.). They do not have a straightforward way to include higher granularities of user feedback, such as ratings on a scale of 1 to 5. The absence of such a technique leads to the formation of our RQ2 that is then addressed in Article PII (Pandey et al., 2019).

2.4 Deep Learning for Recommender Systems

In this section, we briefly describe the efforts in deep learning for recommender systems and explain serendipitous recommender systems.

2.4.1 Deep Learning

Deep learning is receiving massive attention for quite a few years now. It is a type of machine learning method, that learns representations of data at multiple layers that correspond to different levels of abstraction. Because of its effectiveness, deep learning finds application in a wide range of domains. For example, there are many efforts in the areas of computer vision (He et al., 2016; Donahue et al., 2014; Ranjan et al., 2019), natural language processing (Young et al., 2018; Hirschberg and Manning, 2015) and speech recognition (Amodei et al., 2016; Zhang et al., 2017; Deng et al., 2013).

Moreover, there have also been various efforts in the field of recommender systems that utilize the available deep learning techniques. Zhang et al. (2019) provide a recent survey of such deep learning based recommender systems. Some of the recent efforts that have carried out recommendation tasks using deep networks are: He et al. (2017); Cheng et al. (2016); Covington et al. (2016); Wu et al. (2016); Ying et al. (2018).

2.4.2 Serendipitous Recommender Systems

Traditionally, the focus of recommender systems has been focused around the relevance of items that are presented to the user as recommendations. This focus however has been shifting since relevance shall not be considered the sole criteria of the usefulness of the items. For example, if the user is already familiar with the recommended items, then in spite of the items being relevant, they do not offer any new information to the user and hence are not useful. Serendipitous recommender systems have recently come up as a possible solution to this problem (Kotkov et al., 2016). Serendipitous recommendations are often described as the ones that are a combination of relevance, novelty and unexpectedness (Kotkov et al., 2018; McNee et al., 2006). Various efforts support the idea that recommender algorithms should suggest serendipitous items to broaden user preferences and improve their satisfaction (Kotkov et al., 2018; Zhang et al., 2012). There have also been some efforts to create serendipity oriented algorithms (Lu et al., 2012; Zheng et al., 2015; Yamaba et al., 2013; Kotkov et al., 2019).

However, to the best of our knowledge there are no efforts that use deep learning or transfer learning to recommend serendipitous items. This leads to the formation of RQ3 that is addressed in Article PIII (Pandey et al., 2018a)

2.5 Metrics

Lastly, we describe the metrics used for the comparisons carried out in the presented research.

2.5.1 Rank Accuracy Metrics

In the presented work, the accuracy of rankings created by the document ranking systems or the reccommender systems, is considered as the prime indicator of the performance of the underlying algorithm. For this, we have used two standard ranking accuracy metrics: mean average precision (*MAP*) (Baeza-Yates and Ribeiro-Neto, 1999) and normalized discount cumulative gain (*NDCG@n*) (Järvelin and Kekäläinen, 2002).

The metric *MAP* has been used for calculating ranking accuracy of document ranking systems only, in this dissertation. Therefore, we explain it from the perspective of document ranking. Let us consider a set of queries Q, where the

document ranking system returns a ranking of documents for each query. Then, for a particular query q from this query set, precision for its top n results, i.e. $P@n$, can be calculated as:

$$P@n = \frac{\text{\# relevant documents in top } n}{n}. \tag{1}$$

Consider $\text{rel}(k)$ to be a function that maps the document at position k to a binary status of relevant (1) or irrelevant (0). Now, AP_q or Average Precision for the query q, for its top N results, can be calculated by taking the average of the $P@n$ values for all relevant documents:

$$AP_q = \frac{\sum_{k=1}^{N} \big(P@k \times \text{rel}(k)\big)}{\text{\# relevant docs for } q}, \tag{2}$$

Then MAP takes the mean of the average precision values over all queries:

$$MAP = \frac{\sum_{q \in Q} AP_q}{\text{\# queries in } Q}. \tag{3}$$

It should be noted MAP can only consider binary relevance (relevant or irrelevant). It is not able to directly handle relevance in multiple levels, for example if the relevance had been defined as: 0 (irrelevant), 1 (somewhat relevant) and 2 (relevant).

The second metric measure $NDCG@n$, that has been used quite frequently in the presented research, is capable to handle multiple levels of relevance. We have used it for calculating accuracies for document ranking systems as well as for recommender systems. To calculate it lets firstly define $DCG@n$ (discounted cumulative gain) as:

$$DCG@n = \sum_{j=1}^{n} \frac{2^{\text{rel}(j)} - 1}{\log(j+1)}, \tag{4}$$

where $\text{rel}(j)$ can have non-binary values. For instance, in document ranking system, it can be a relevance level of 0, 1 or 2, for a document with respect to the query. Moreover, for ranking of items in recommender systems, these could be the ratings given by the user on a scale (e.g. 1 to 5 star ratings for movies or products). Now, $NDCG@n$ can simply be calculated as:

$$NDCG@n = \frac{DCG@n}{IDCG@n}, \tag{5}$$

where $IDCG@n$ is the ideal discounted cumulative gain that is possible. For multiple users in recommender system or multiple queries in document ranking system, the $NDCG$ is averaged to form the final metric.

2.5.2 Feature Quality Metrics

To measure the quality of the features extracted for document ranking, in this research, we consider two metrics: importance and redundancy.

The value of each feature would typically be different for different documents. Now, the *importance* of a particular feature can be estimated by calculating the ranking performance achieved, when only that feature is used as a ranking model to order documents. For this, we can use a ranking accuracy measure, e.g. $NDCG@n$ for a particular value of n. While calculating these measures, larger values for some features might correspond to higher rank of the document, but for other features higher values might correspond to lower rank. Therefore, we utilize the strategy used by GAS (Geng et al., 2007) for evaluation. In this strategy, the documents are ordered two times: in ascending order and in descending ordered. Then, we take the bigger score as the importance score of the feature. Once we have the importance scores for all the features from the feature set $F = \{f_1, f_2, \ldots, f_k\}$, the average importance is calculated as:

$$\text{Imp}(F) = \frac{1}{k}\sum_{i=1}^{k} \max\left\{ \text{eva}(\mathcal{X}, f_i), \text{eva}(\mathcal{X}, -f_i)\right\}, \tag{6}$$

where the function $\text{eva}(\mathcal{X}, f_i)$ returns the value of ranking evaluation, when f_i is used as a ranking model on the dataset $\mathcal{X}$. A higher value of $\text{Imp}(F)$ indicates, better quality of the features.

The *redundancy* of features can be calculated using the similarities between each pair of features and then taking their average. For this, we can consider feature as a ranking model, and use it to order the documents. Now, the similarity between a pair of features can be calculated as the average of their document ranking similarities for different queries. Let us consider Q to be a given set of queries and $F = \{f_1, f_2, \ldots, f_k\}$ to be the features of the documents. Also, for a particular query q from Q, let $\sigma_i^{(q)}$ be the ranking of the documents when the feature f_i alone is used as the ranking model to rank the documents. Now, the redundancy of the features F is calculated as:

$$\text{Rdd}(F) = \frac{2}{k(k-1)} \sum_{f_i, f_j \in F, i>j} \frac{1}{|Q|} \sum_{q \in Q} \text{sim}\left(\sigma_i^{(q)}, \sigma_j^{(q)}\right), \tag{7}$$

where $\text{sim}\left(\sigma_i^{(q)}, \sigma_j^{(q)}\right)$ is the similarity between the rankings $\sigma_i^{(q)}$ and $\sigma_j^{(q)}$. To calculate this similarity, we use the absolute value of Kendall's τ correlation coefficient (Kendall, 1948). Considering N_c and N_d as the numbers of the concordant pairs and discordant pairs respectively between the rankings σ_i and σ_j, the Kendall's τ correlation coefficient is calculated as:

$$\tau\left(\sigma_i, \sigma_j\right) = \frac{N_c - N_d}{N_c + N_d}, \tag{8}$$

The value of $\tau\left(\sigma_i, \sigma_j\right)$ lies between -1 and 1. The negative and positive values of this coefficient indicate negative correlation and positive correlation, respectively. The magnitude of the value indicates how strong the correlation is and a value of 0 indicates no correlation. Since the strength of correlations is important and it is not desired that the negative and positive values cancel each other's effect on

averaging, we consider the absolute value of Kendall's τ as the similarity metric. In other words, we calculate $\text{sim}\left(\sigma_i^{(q)}, \sigma_j^{(q)}\right) = \left|\tau\left(\sigma_i^{(q)}, \sigma_j^{(q)}\right)\right|$. A lower value of $\text{Rdd}(F)$ indicates lesser redundancy and hence better quality of the features.

2.5.3 Vector Quality Metrics

RMSE (root mean squared error) and MAE (mean absolute error) are the metrics used by us to calculate the quality of vectors. We use them specifically in the context of recommender systems to measure the quality of item vectors. Given n items, with item vector for an item i denoted as $\boldsymbol{I}_i$ and the corresponding ground truth denoted as $\boldsymbol{G}_i$, the error for a pair of items i and j is calculated as:

$$e_{i,j} = \text{vecSim}(\boldsymbol{I}_i, \boldsymbol{I}_j) - \text{gtSim}(\boldsymbol{G}_i, \boldsymbol{G}_j), \tag{9}$$

where the functions vecSim() and gtSim() calculate the similarities between a pair of vectors and ground truths, respectively. Moreover, the items in our research are movies from the Movielens datasets [1] and the genres to which each movie belongs to are also available to us. This allows us to use this as the ground truth. In a different setup, different ground truth could be used.

Also, a movie can belong to one or more genres, and its state corresponding to each genre can be considered as a binary value. This means that a value of 1 for a particular genre signifies that the movie belongs that genre and a value 0 signifies the opposite. This leads to the formation of genre vectors containing binary values for each movie. Therefore we could consider the ground truth $\boldsymbol{G}_i$ as the genre vector corresponding to the movie i (i.e. item in our case). Now the similarity between the item vectors $\text{vecSim}(\boldsymbol{I}_i, \boldsymbol{I}_j)$ can be calculated using cosine similarity. Unlike the item vectors, the genre vectors are composed of only binary values. Therefore, similarity between the genre vectors $\text{gtSim}(\boldsymbol{G}_i, \boldsymbol{G}_j)$ is calculated using Jaccard similarity (Choi et al., 2010), a popular and efficient binary similarity measure. The main idea behind using the differences of these similarities as the error is that two movie vectors with similar genres should also have similar vector representations.

This enables us to calculate our quality metrics RMSE and MAE as:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n} e_{i,j}^2}{n(n-1)/2}}, \tag{10}$$

$$\text{MAE} = \frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n} |e_{i,j}|}{n(n-1)/2}, \tag{11}$$

where the function $|\cdot|$ gives the corresponding absolute value. The lower values of RMSE and MAE metrics would indicate better quality of item vectors.

[1] https://grouplens.org/datasets/movielens/

3 KEY CONTRIBUTIONS

In this chapter we describe the main contributions and explain how they answer our research questions.

3.1 Linear Feature Extraction for Ranking

Our first main contribution is to answer RQ1 by the creation of *LifeRank* algorithm from Article PI (Pandey et al., 2018b), a feature extraction algorithm for learning to rank.

LifeRank generates a low dimensional dataset $\mathcal{X}'$ for a given high dimensional dataset $\mathcal{X}$. In the first phase, the training dataset $\mathcal{X}$ is preprocessed to create an original matrix $\boldsymbol{X}$ and other information $\boldsymbol{I}_X$. Here $\boldsymbol{X}$ consists of m rows corresponding to the documents in the training set and n columns represent features for the documents. Moreover, $\boldsymbol{I}_X$ consists of relevance labels, ids of the documents and queries, etc. Now let us consider $D = \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_m\}$ to be the document vectors in the matrix $\boldsymbol{X}^{\top}_{m \times n}$. We utilize each each pair of document vectors $(\boldsymbol{d}_i, \boldsymbol{d}_j) \in D \times D$ to create the pairwise labels for the training dataset, i.e. $y_{i,j} \in \{+1, -1\}$. The value of $y_{i,j}$ is $+1$ when the relevance of the i-th document is higher than the j-th document for a query. The value is -1 in the opposite case. Moreover, we initialize a transformation matrix $\boldsymbol{T}_{n \times k}$ with $\{\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_k\}$ as its column vectors. $\boldsymbol{T}$ needs to be optimized, so that it can be used to transform $\boldsymbol{X}$ with n features to $\boldsymbol{X}'$ with k features, where $k < n$, as shown in Figure 1.

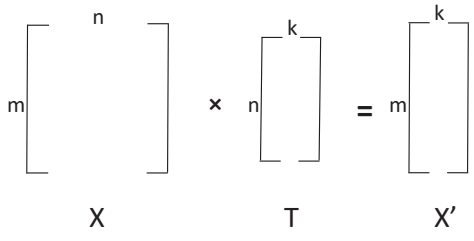


FIGURE 1 Transformation of X to lower dimension

LifeRank aims to minimize the loss function in Equation 12 from Pandey et al. (2018b) in order to optimize $\boldsymbol{T}$. For this, we also initialize $\boldsymbol{A}$, a matrix with k columns and k rows, and elements $\alpha_{i,j}$; as well as w, a k-dimensional vector of weights and a constant b:

$$\mathcal{L}(\boldsymbol{T}, \boldsymbol{w}, b, \boldsymbol{A}) = \sum_{\forall(\boldsymbol{d}_i, \boldsymbol{d}_j), i \neq j} \log\left(1 + e^{-y_{i,j}\left(\boldsymbol{w}^\top \boldsymbol{T}^\top (\boldsymbol{d}_i - \boldsymbol{d}_j) + b\right)}\right) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{i,j=1,\ldots,k \wedge i \neq j} \alpha_{i,j} \boldsymbol{t}_i^\top \boldsymbol{t}_j + \sum_{i=1}^{k} \alpha_{i,i}\left(1 - \boldsymbol{t}_i^\top \boldsymbol{t}_i\right). \quad (12)$$

Here the first part of the loss function calculates the log loss of the ranking accuracy. Second part is introduced to avoid overfitting, where $\|\cdot\|^2$ represents the L2 norm and λ is the coefficient of the regularization. The last two parts in the loss function are there to impose orthonormality constraints on $\boldsymbol{T}$. The detailed motivation for the loss function is explained in Article PI (Pandey et al., 2018b).

To optimize the loss function, the gradients with respect to the variables w, t_l, b and $\alpha_{i,j}$ are calculated and then used to run gradient descent. In the second phase, *LifeRank* proceeds to generate low-dimensional training, validation and test matrices using the optimized transformation matrix $\boldsymbol{T}$. Then *LifeRank* forms new datasets based on these low-dimensional data matrices by combining them with the previously removed extra information. The algorithm that is published in the article Pandey et al. (2018b), is shown verbatim in Algorithm 1.

The number of documents is expected to be highly influential for the complexity of the algorithm. As we can see that the loss function and also the gradients consider the pairs of documents, even a small increase in number of documents would increase the running time considerably.

The *LifeRank* algorithm was used for dimensionality reduction on the LETOR[1] datasets that are the common benchmarks for learning to rank. We used MQ2007 and MQ2008 datasets from LETOR 4.0 (Qin and Liu, 2013) and OHSUMED from LETOR 3.0 (Qin et al., 2010). The datasets created after feature extraction were evaluated for learning to rank algorithms: RankSVM (Joachims et al., 2009) and Linear Regression (Lawson and Hanson, 1995). The evaluations, presented in detail in Article PI (Pandey et al., 2018b), showed improvements over standard feature selection techniques: GAS (Geng et al., 2007) and FSMSVM (Lai et al., 2013).

3.2 Vectors of Pairwise Item Preferences

Our second contribution is to answer RQ2 by the creation of *Pref2Vec* algorithm from Article PII (Pandey et al., 2019), a method to generate vectors of pairwise item preferences. The preference vectors are then used to create user vectors and item vectors.

[1] http://research.microsoft.com/en-us/um/beijing/projects/letor/

Algorithm 1: *LifeRank*: A Linear Feature Extraction Algorithm for Ranking (verbatim from Pandey et al. (2018b))

Input: A training dataset $\mathcal{X}$, a validation dataset $\mathcal{V}$, a test dataset $\mathcal{E}$, the learning rate η, and the number of features k in the set of generated document.

Output: A generated training dataset $\mathcal{X}'$, validation dataset $\mathcal{V}'$, and test dataset $\mathcal{E}'$, each with k features.

// Phase I

$(\boldsymbol{X}, \boldsymbol{I}_X) \leftarrow$ Preprocess$(\mathcal{X})$;

$\boldsymbol{T}, \boldsymbol{w}, \{\alpha_{i,j}\}_{i,j=1,\ldots,k} \leftarrow$ Initialize$(\boldsymbol{X}, k)$;

repeat

  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} \mathcal{L}$;

  $\boldsymbol{t}_l \leftarrow \boldsymbol{t}_l - \eta \nabla_{\boldsymbol{t}_l} \mathcal{L}$, for $l = 1, \ldots, k$;

  $b \leftarrow b - \eta \frac{\partial \mathcal{L}}{\partial b}$;

  $\alpha_{i,j} \leftarrow \alpha_{i,j} + \eta \frac{\partial \mathcal{L}}{\partial \alpha_{i,j}}$, for $i, j = 1, \ldots, k$;

until *Reach convergence or the max iteration*;

// Phase II

$(\boldsymbol{V}, \boldsymbol{I}_V), (\boldsymbol{E}, \boldsymbol{I}_E) \leftarrow$ Preprocess$(\mathcal{V}, \mathcal{E})$;

$\boldsymbol{X}' \leftarrow \boldsymbol{X}\boldsymbol{T}, \quad \boldsymbol{V}' \leftarrow \boldsymbol{V}\boldsymbol{T}, \quad \boldsymbol{E}' \leftarrow \boldsymbol{V}\boldsymbol{T}$;

$\mathcal{X}', \mathcal{V}', \mathcal{E}' \leftarrow$ GenerateDatasets$(\boldsymbol{X}', \boldsymbol{V}', \boldsymbol{E}', \boldsymbol{I}_X, \boldsymbol{I}_V, \boldsymbol{I}_E)$;

Let us consider a set of m users: $\{u_1, u_2, \ldots, u_m\}$ as well as a set of n items: $\{I_1, I_2, \ldots, I_n\}$. Also, consider their rating matrix $R_{m \times n}$, where each row of R consists of ratings $R_u = \{r_1, r_2, \ldots, r_n\}$ given by a user u, while it is expected that various ratings could be missing. Now the *Pref2Vec* framework builds a set of pairwise preference for each user. For this, it uses a preference function: $p(i, j) \in \{+1, -1\}$, where $i, j = 1 \ldots n$, $i \neq j$ and both the ratings r_i and r_j are known. The preference value $p(i, j)$ is equal to $+1$ if $r_i > r_j$. Otherwise the value is -1. Using this, *Pref2Vec* creates the sets of positive preferences P_u for each user u. For each user, the preferences in its set would be a subset of $P = \{p_1, p_2, \ldots, p_N\}$ where $N = n(n-1)$.

Example 1 Let us consider that for the items $I = \{I_1, I_2, I_3, I_4\}$. A user has given the ratings $1, 5, 4$ *and* 2 respectively. Here the set of preferences for the user would be $P_u = \{I_2 \succ I_1, I_2 \succ I_3, I_2 \succ I_4, I_3 \succ I_1, I_3 \succ I_4, I_4 \succ I_1\}$. To clarify, the preferences $I_2 \succ I_1$ and $I_2 \succ I_3$ exist in this set because the values of the preference functions $p(2, 1)$ and $p(2, 3)$ both are $+1$, i.e. the ratings indicate that the user prefers I_2 over I_1 as well as I_2 over I_3.

Now, *Pref2Vec* proceeds with learning the vector representations of the preferences on the collection of preference sets $\mathbb{P} = \{P_1, P_2, \ldots, P_m\}$ for all of the users. We utilize the *word2vec* framework (Mikolov et al., 2013a,b) that considers a text corpus and generates vector representations of words. In our approach we similarly consider $\mathbb{P}$ as the corpus, the preference sets $P_1, P_2, \ldots, P_m$ by the users as the

sentences and the preferences $p_1, p_2, \ldots, p_N$ as the words. The main difference in our approach is that we ignore the sequential information in the preference sets. This is because, we consider a non-temporal setup where we do not consider the time when the users had given their ratings. The sequence of preferences created by our approach is arbitrary and hence inconsequential. For example, it should not make a difference if the same three preferences are mentioned in the sequence: $I_2 \succ I_1, I_2 \succ I_3, I_1 \succ I_4$ or in the sequence: $I_2 \succ I_3, I_1 \succ I_4, I_2 \succ I_1$. Our approach is inspired by *item2vec* (Barkan and Koenigstein, 2016) that creates the vectors of items. We however create the vectors of preferences, i.e. corresponding to the preferences $p_1, p_2, \ldots, p_N$, we get the preference vectors $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N$.

Once the preference vectors are created, the next step in our approach is to create user vectors using them. Let us consider for a particular user: $p_1, p_2, \ldots, p_r \in \{+1, -1\}$ to be the preference values and $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_r$ to be the corresponding preference vectors of length k. To create the corresponding user vector $\boldsymbol{u}$ also of length k, the approach tries to find such a vector so that cumulatively each preference p_i matches with the corresponding product of the user and preference vector $\boldsymbol{u}^\top \boldsymbol{p}_i$. In order to find optimal vector $\boldsymbol{u}$, we train a linear classification model that uses Logistic regression (Hosmer Jr et al., 2013) to optimize the following loss function where the second part is regularization term with L2 norm:

$$\arg\min_{\boldsymbol{u},b} \sum_{i=1}^{r} \log(1 + \exp(-p_i(\boldsymbol{u}^\top \boldsymbol{p}_i + b))) + \frac{\lambda}{2}\|\boldsymbol{u}\|^2, \tag{13}$$

where b is a number and λ is a tuning parameter. It should be noted that the optimization of this loss function is expected to create user vector $\boldsymbol{u}$, such that $\boldsymbol{u}^\top \boldsymbol{p}_i$ has a high value for $\boldsymbol{p}_i = +1$ and a low value for $\boldsymbol{p}_i = -1$.

For the loss function we calculate the gradients with respect to the variables $\boldsymbol{u}$ and b, and use them to run gradient descent method for optimization. The user vector generation is shown in Algorithm 2. The generated user vectors $\boldsymbol{u}$ corresponding to each of the m users and each of length k can also be assembled to form a user matrix $U_{m \times k}$.

Algorithm 2: User Vector Generation

Input: For a user: preference vectors $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_r$, with respective ground truth values $p_1, \ldots p_r \in (+1, -1)$, and learning rate α

Output: Optimized values of $\boldsymbol{u}$ and b

1 *Initialize* $\boldsymbol{u}$ *and* b;

2 **repeat**

3
$$\boldsymbol{u} \leftarrow \boldsymbol{u} - \alpha\left(\sum_{i=1}^{r} \frac{-p_i}{1 + \exp(p_i(\boldsymbol{u}^\top \boldsymbol{p}_i + b))}\boldsymbol{p}_i + \lambda \boldsymbol{u}\right)$$
$$b \leftarrow b - \alpha\left(\sum_{i=1}^{r} \frac{-p_i}{1 + \exp(p_i(\boldsymbol{u}^\top \boldsymbol{p}_i + b))}\right)$$

4 **until** *reach convergence or the max iteration*;

5 **return** $\boldsymbol{u}$, b

The final step of *Pref2Vec* is to create item vectors. For this it utilizes the previously created user matrix $U_{m\times k}$ along with the given rating matrix $R_{m\times n}$. The approach tries to create an item matrix $I_{n\times k}$ with each row corresponding to an item vector. This is done by minimizing the difference between observed ratings in R and the product of user and item vectors i.e. $UI^\top \approx R$. For this, we minimize the loss function:

$$\arg\min_I \|R - UI^\top\|^2 + \frac{\lambda}{2}\|I\|^2, \tag{14}$$

where λ is the parameter for the L2 norm. The gradient of the loss function with respect to the variable I is calculated and then it is optimized using gradient descent, as shown in Algorithm 3.

Algorithm 3: Item vectors derivation

Input: User matrix U of dimension $m \times k$, rating matrix R of dimension $m \times n$ and learning rate η

Output: Optimized item matrix I of dimension $n \times k$

```
Initialize I;
repeat
    I ← I − η(−2(R − UI^⊤)^⊤U + λI)
until Reach convergence or the max iteration;
return I
```

It should be noted that the preference vector creation in *Pref2Vec* works similar to *item2vec*, and hence similar complexity is expected. However, the main difference is that the number of positive preference pairs for a user, can typically be larger than the items. Moreover, in certain cases where the user has given the same ratings to all the items, no preference pairs would be formed.

The *Pref2Vec* algorithm was used on benchmark movie recommendation datasets from MovieLens [2]: MovieLens-100K, MovieLens-1M and MovieLens-10M. The offline evaluations showed that the generated item vectors were better in quality than the standard techniques including *item2vec* (Barkan and Koenigstein, 2016). Moreover, the user vectors as well as item vectors were shown to be independent of initializations. Also, the utility of the algorithm was shown by using the created vectors for predicting recommendation rankings, where our technique outperformed the standard recommendation techniques. The detailed results are presented in Article PII (Pandey et al., 2019).

3.3 Recommending Serendipitous Items using Transfer Learning

Our third contribution is to answer RQ3 by the creation of *SerRec* algorithm from Article PIII (Pandey et al., 2018a), a transfer learning technique to recommend

[2] http://grouplens.org/datasets/movielens/

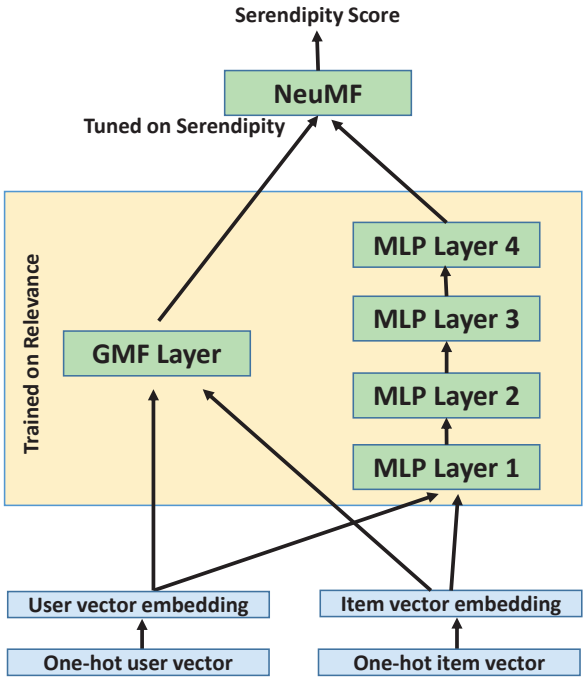


FIGURE 2 *SerRec* Architecture for Transfer Learning for Serendipity. Architecture uses NCF framework (He et al., 2017) and the figure is from Pandey et al. (2018a)

serendipitous items. Since, there are no large datasets available with serendipity scores, it firstly trains a deep neural network on a large dataset with relevance scores and then tunes it using a smaller dataset with serendipity scores.

For this we utilized the Neural Collaborative Filtering (NCF) framework[3] introduced by He et al. (2017). Our transfer learning technique that utilizes this is shown in Figure 2. As it can be seen in the figure, the NCF framework is composed of two parts: Generalised Matrix Factorization (GMF) and a group of four Multi Layer Perceptron (MLP) dense layers. While this framework is originally used for predicting the relevance scores, we employ transfer learning for predicting serendipity scores.

For this, we first train the GMF and MLP layers separately, using a large dataset with relevance scores. Thereafter, we fix the weights in these trained layers and then tune the NeuMF layer using a smaller serendipity oriented dataset. For our offline evaluations, we employed Serendipity-2018 dataset (Kotkov et al., 2018), that to the best of our knowledge is the only publicly available dataset with serendipity scores provided by the users. This dataset contains a large dataset with relevance scores along with a smaller dataset with serendipity as well as relevance scores. The bigger dataset was used to train on relevance, and then the smaller data was utilized for tuning and testing on serendipity. Our approach to recommend rankings of serendipitous items outperformed the comparison algorithms that included Singular Value Decomposition (Koren and Bell, 2015), Serendipitous Personalized Ranking (Lu et al., 2012) and Unexpectedness-

[3] https://github.com/hexiangnan/neural_collaborative_filtering

Augmented Utility Model (Zheng et al., 2015). The detailed results can be seen in Article PIII (Pandey et al., 2018a).

3.4 Listwise Recommendation Approach with Non-negative Matrix Factorization

Our fourth contribution is to answer RQ4 by the creation of *LwRec* algorithm from Article PIV (Pandey and Wang, 2018). *LwRec* is a novel listwise ranking-oriented matrix factorization algorithm, that instead of using a rating matrix utilizes a user-ranking probability matrix to predict users' rankings of items.

Let us consider m users and n items. This leads to $n!$ possible rankings of items for each user, which is a very large number even for not so big values of n. *LwRec* uses the approach by Huang et al. (2015) to calculate the probabilities of the rankings of items with known ratings. The approach uses an efficient method by Cao et al. (2007), in which the focus is only on the top-k items in the rankings. This leads to the reduction number of rankings to $\frac{n!}{(n-k)!}$. Using this approach, the probability of the rankings ρ_S having $S = \{i_1, i_2 \dots i_k\}$ as the exact top-k items, is calculated as:

$$Prob(\rho_S) = \prod_{j=1}^{k} \frac{\gamma(r_{i_j})}{\sum_{l=j}^{n} \gamma(r_{i_l})}, \tag{15}$$

where r_{i_j} is the given rating for the item i_j and $\gamma(r) = e^r$.

Example 2 Consider a simple example where a user has given ratings of 2, 3 and 5 to items i_1, i_2 and i_3 respectively. If $k = 2$, we will have $\frac{3!}{(3-2)!} = 6$ different rankings i.e. $\{i_1, i_2\}$, $\{i_1, i_3\}$, $\{i_2, i_1\}$, $\{i_2, i_3\}$, $\{i_3, i_1\}$ and $\{i_2, i_2\}$. Now, using the formula (15) for the probability,

$$Prob(\{i_1, i_2\}) = \frac{e^2}{e^2 + e^3 + e^5} \cdot \frac{e^3}{e^3 + e^5} = 0.005.$$

Similarly,

$$\begin{aligned} &Prob(\{i_1, i_3\}) = 0.037, \quad Prob(\{i_2, i_1\}) = 0.005, \\ &Prob(\{i_2, i_3\}) = 0.109, \quad Prob(\{i_3, i_1\}) = 0.227, \\ &Prob(\{i_3, i_2\}) = 0.617. \end{aligned}$$

It can be seen that the best possible ranking i.e. $\{i_3, i_2\}$ (having the highest rated item on first position and second highest rated on the second position) gets the highest probability.

Since there are m users and $p = \frac{n!}{(n-k)!}$ possible ranking sets for each user; we create $\Theta_{m \times p}$, the user-ranking probability matrix. In Θ, each row contains the probabilities of the p rankings for a particular user. It should be noted that the probability of a ranking for a user would be present in Θ only if all the ratings for the items in that ranking are known for the user. The value otherwise would be

unknown. The sum of probabilities in each row is 1 and the unknown values are denoted by $\perp$.

Moreover, *LwRec* deals with the case of $k = 1$, i.e. it considers only the top items in the rankings. Since $p = \frac{n!}{(n-1)!} = n$ now, the size of Θ would have same dimension as the rating matrix i.e. $m \times n$.

In order to predict the unknown probabilities in Θ, *LwRec* employs two matrices $U_{z\times m}$ and $G_{z\times p}$ to form the predicted probability matrix $U^\top G$, where z can be chosen as a parameter. To get optimum matrices U and G, the *LwRec* approach aims to minimize the following listwise loss function (Pandey and Wang, 2018):

$$L(U, G, \alpha, \beta) = -\sum_{i=1}^{m} \sum_{j=1,\Theta_{ij}\neq\perp}^{p} \Theta_{ij} \log \frac{(U^\top G)_{ij}}{\sum_{l=1,\Theta_{il}\neq\perp}^{p} (U^\top G)_{il}} + \sum_{i=1}^{m} (\alpha_i - \beta_i) \left(\sum_{j=1}^{p} (U^\top G)_{ij} - 1 \right) + \frac{\lambda_1}{2} \|U\|^2 + \frac{\lambda_2}{2} \|G\|^2. \quad (16)$$

Here, $\|\cdot\|^2$ is the L2 norm and λ_1 and λ_2 are the coefficients. The second part of the loss function also considers that since the rows of $U^\top G$ contain probabilities of the rankings, the sum of the values in each row should be 1. Here, α and β are vectors, so that $\alpha_i - \beta_i$ acts as the coefficient for the i^{th} penalty term. Moreover, since the values of probabilities cannot be negative, the approach follows non-negative matrix factorization (Lee and Seung, 2000) to derive the update rules to minimize the loss function. This requires non-negative initialization of all the parameters i.e. U, G, α and β. Since the values contained in any of these will never be negative during optimization because of non-negative matrix factorization, the values in $U^\top G$ will also be non-negative. The reason behind using two vectors α and β is that although neither of them can have negative values, the effective coefficient $\alpha_i - \beta_i$ can be negative as well as non-negative. The detailed motivation as well as technique to minimize the loss function is presented in Pandey and Wang (2018).

Upon optimization of the loss function, the finally created $U^\top G$ matrix contains the predictions of ranking probabilities. Since we have considered $k = 1$, each row of $U^\top G$ would contain probabilities for n one-item rankings. Therefore, items can be ordered by their decreasing predicted probabilities in order to form a ranking.

Also, as mentioned we use k = 1, so the size of Θ is $m \times n$. However, increasing the value of k is expected to lead to a considerable increase in the number of columns in Θ, and hence increasing the complexity of the algorithm. The *LwRec* algorithm used benchmark movie recommendation datasets from MovieLens: MovieLens-100K and MovieLens-1M. The offline evaluations showed that the *LwRec* algorithm outperformed the state-of-the-art recommendation techniques. The detailed results are presented in Article PIV (Pandey and Wang, 2018).

3.5 Utilization of Neural Embeddings for finding Cities and Tours

Our fifth contribution is to answer RQ5 by the creation of *CitySearcher* algorithm from Article PV (Abdel Maksoud et al., 2017) and tour generation algorithm from Article PVI (Abdel Maksoud et al., 2018).

3.5.1 CitySearcher

CitySearcher algorithm is designed to return a ranking of cities when queried for travel interests (e.g. music, nightlife, beach, etc.). The algorithm uses *word2vec* (Mikolov et al., 2013b,a) on a corpus in order to calculate the vectors corresponding to all the words present in the corpus vocabulary. The documents contained in the corpus are predominantly about cities, where each document represents one particular city and each city is represented by only one document.

For ranking the cities in response to an interest *itr* i.e. a word given by the user expressing her travel interest, the algorithm finds the ranking score for the city documents for it. To find this score for a city c, the similarity scores are calculated between the vector for *itr* and the vectors of words present in the city document. Now considering the highest k such scores: $s_1, s_2 ... s_k$, the ranking score, initialScore(c, itr) can be formulated as:

$$\text{initialScore}(c, itr) = \frac{1}{k} \sum_{i=1}^{k} s_i. \tag{17}$$

For example, if the top 5 similarity scores between an interest and words in a city document are: 0.9, 0.8, 0.7, 0.5 and 0.4, then for $k = 3$ the resultant score would be calculated as: $\frac{0.9+0.8+0.7}{3} = 0.8$. On calculating such score for each city, the ranking of the cities can be achieved by sorting them in a decreasing order of the scores.

This approach produces decent rankings, but in certain cases, it suffers from the problem of incorrect semantic mismatching. For example, for the travel interest *romance*, it awards high ranking to cities with frequent mention of *Roman* artifacts. Moreover, the city name is semantically interpreted. Due to this, it leads to incorrect interpretation by the algorithm, when a city called *Sale* is ranked first for the interest *Shopping*.

To improve the algorithm and remove such issues, the algorithm attempts to create novel features for the documents and train models using machine learning algorithms such as Kernel ridge regression (Murphy, 2012) and Logistic regression (Hosmer Jr et al., 2013). The training is done using relevance assessments obtained from the users.

CitySearcher used the English version of Wikivoyage dataset [4] for our experiments. The documents contained in the dataset predominantly represent different cities. A list of words representing common interests was considered for the

[4] https://en.wikivoyage.org/wiki/Main_Page

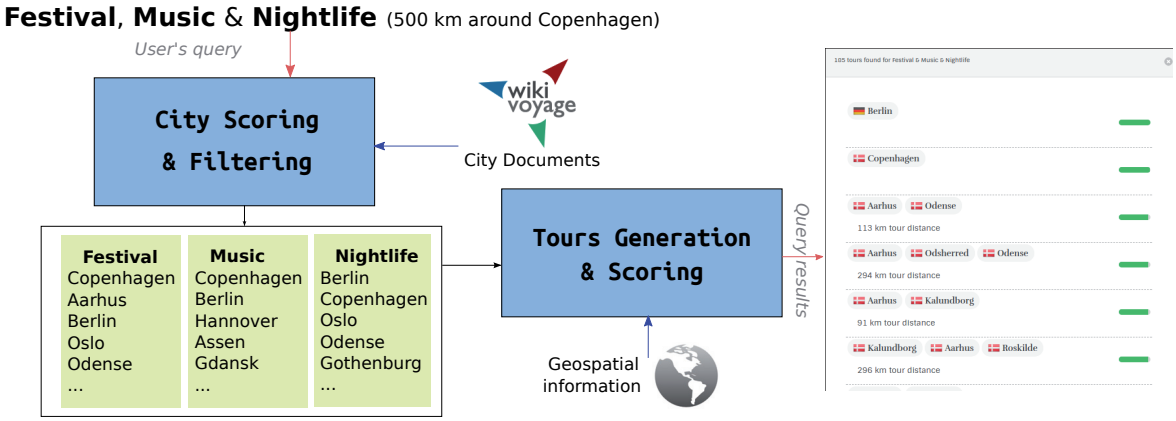


FIGURE 3 Architecture of *Travición*. The diagram is from Abdel Maksoud et al. (2018).

experiments. Ratings for city-interest pairs, that were collected using the crowdsourcing platform clickworker [5], were used to create training and test sets. In our results, *CitySearcher* algorithm shows improvements over standard document ranking techniques. The detailed results are presented in Article PV (Abdel Maksoud et al., 2017).

3.5.2 Tour Generation

Moreover, our tour generation technique to find ranking of *tours* (or groups of cities) in response to a *set of travel interests*. Given a set of interests $\{itr_1 \dots itr_n\}$, the objective is to create a ranking of tours $\langle t_1, \dots t_m \rangle$, where each t_i contains one or more cities. The technique aims that each interest in the interest set is satisfied by at least one city in the tour. Moreover, it is aimed that the cities in the tour are not too far apart, that would make the tour impractical for the users. For this, we utilize the already available rankings of city documents for individual interests from *CitySearcher*. For forming the tour scores, we explored novel tour scoring techniques. These scoring techniques and their comparisons are presented in detail in Article PVI (Abdel Maksoud et al., 2018). The main outcome is the creation and demonstration of the *Travición* website [6]. Figure 3 describes the architecture of *Travición*.

5 https://clickworker.com
6 https://www.travicion.com/

4 SUMMARY OF ARTICLES

This chapter summarizes the original articles that are included in this dissertation. For each article, the aim is explained along with the achieved results and the author's specific contribution.

4.1 Article PI: "Linear feature extraction for ranking"

Gaurav Pandey, Zhaochun Ren, Shuaiqiang Wang, Jari Veijalainen and Maarten de Rijke. Linear feature extraction for ranking. Information Retrieval Journal 21(6): 481-506, 2018.

Aim The article addresses the problem of linear feature extraction for learning to rank. For this, we propose a novel and efficient algorithm that uses the existing original features of documents in a dataset and extracts features in lower dimension. These extracted features are a linear combination of original features and are aimed to be used as an input to learning to rank algorithms (for document ranking). This is novel because, although there are various feature selection algorithms for learning to rank, to the best of our knowledge there are no feature extraction algorithms for this.

Results We proposed our algorithm *LifeRank* (Linear Feature Extraction for Ranking) that considers each dataset (training, validation or test) as original matrix, where each row of the matrix represents a document and consists of its original features. For a original training matrix $\mathbf{X}$, the algorithm aims to find a transformation matrix $\mathbf{T}$, so that by multiplying these two, $\mathbf{X}$ is projected into a lower dimensional matrix $\mathbf{X'}$, i.e. $\mathbf{X' = XT}$. To optimize the transformation matrix $\mathbf{T}$, *LifeRank* minimizes a pairwise loss function, since such a loss function is fundamental, straightforward and intuitive to ranking. Once we get the optimized $\mathbf{T}$, it can simply be used to transform the training, validation and test sets. The extensive experiments on benchmark datasets and standard learning to rank algo-

rithms show performance gains of *LifeRank* over state-of-the-art feature selection algorithms.

Author's Contribution The author took active part in problem formulation, development of the algorithm and writing the initial daft of paper. The author also implemented the software for the proposed algorithm and conducted the evaluations presented in the article. The final version of the paper was created by the collaboration and inputs of all the authors.

4.2 Article PII: "Vectors of pairwise item preferences"

Gaurav Pandey, Shuaiqiang Wang, Zhaochun Ren and Yi Chang. Vectors of pairwise item preferences. In European Conference on Information Retrieval, pp. 323-336. Springer, 2019.

Aim The article addresses the problem of using neural embedding to utilize users' explicit feedback (e.g. ratings) on items, to generate good quality user vectors and item vectors. The existing neural embedding algorithms only consider whether the user has accessed the items, but do not consider whether the user has given a high or low rating to the item.

Results We proposed our algorithm *Pref2Vec* that creates vectors of pairwise item preferences. For this, it uses neural embedding and considers users' pairwise preferences on items as elementary units. These pairwise preference vectors are then used to generate user and item vectors, that are representative of the user and item respectively. The experimental results show that the quality of the item vectors is superior to the standard vectorization techniques. Also, we show the initialization independence of the user and item vectors. Moreover, for demonstrating utility in a recommendation task, we generate the ranking of items using the user vectors and show the improvements over standard recommendation techniques.

Author's Contribution The author took active part in problem formulation, development of the algorithm and writing the initial daft of paper. The author also implemented the software for the proposed algorithm and conducted the evaluations presented in the article. The final version of the paper was created by the collaboration and inputs of all the authors.

4.3 Article PIII: "Recommending serendipitous items using transfer learning"

Gaurav Pandey, Denis Kotkov and Alexander Semenov. Recommending serendipitous items using transfer learning. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management, pp. 1771-1774. ACM, 2018.

Aim The article aims to use deep neural networks for recommending serendipitous items, since they have not been explored much for serendipitous item recommendations. Moreover, recommendation of serendipitous items suffers from the lack of large datasets containing serendipity scores. Therefore, we also aim to use transfer learning to overcome this.

Results In this article we introduce *SerRec*, a transfer learning algorithm to recommend serendipitous items. While relevance scores are available to us in abundance, the availability of serendipity scores is very limited. Therefore, the algorithm uses transfer learning and trains a deep neural network using the relevance score. Later, it tunes the network using the serendipity dataset. The experimental results show improvements over the state-of-the-art serendipity oriented baseline algorithms.

Author's Contribution The author took active part in problem formulation, development and software implementation of the algorithm, conducting the experiments and writing the initial daft of paper. The final version of the paper was created by the collaboration and inputs of all the authors.

4.4 Article PIV: "Listwise recommendation approach with non-negative matrix factorization"

Gaurav Pandey and Shuaiqiang Wang. Listwise recommendation approach with non-negative matrix factorization. In Intelligent Decision Technologies 2018, pp. 22-32. Springer, Cham, 2018.

Aim Most of the known ranking oriented matrix factorization algorithms generate recommendations on the basis of the rating matrix. In order to utilize the users' preferred ranking on items, the article aims to propose an algorithm that creates and utilizes a user-ranking probability matrix, i.e. a matrix where each row contains probabilities of item rankings for a particular user.

Results The article proposes *LwRec*, a listwise ranking oriented matrix factorization algorithm, that predicts the missing values in the original user-ranking probability matrix. For this, it considers that the rows of the predicted matrix should have probability distributions similar to the initial matrix. The recommendations generated by *LwRec* show improved performances over baselines.

Author's Contribution The author took active part in problem formulation, development of the algorithm and writing the initial daft of paper. The author also implemented the software for the proposed algorithm and conducted the evaluations presented in the article. The final version of the paper was created by the collaboration and inputs of all the authors.

4.5 Article PV: "CitySearcher: A city search engine for interests"

Mohamed Abdel Maksoud, Gaurav Pandey (first co-author) and Shuaiqiang Wang. CitySearcher: A city search engine for interests. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1141-1144. ACM, 2017.

Aim The article aims to propose a vertical search engine for finding cities for a travel interest, while solving the issue of mismatched semantic relationships.

Results The article introduces *CitySearcher*, a vertical search engine that searches cities for an interest, from a dataset consisting of documents that represent one city each. *CitySearcher* uses *word2vec* to generate the embeddings of the words to generate the initial ranking of cities in response to interests. Moreover, to solve the issue of mismatched semantic relationships, we generate novel features that are used by learning to rank algorithms to re-rank the cities. Our algorithm shows improvements over the standard retrieval techniques.

Author's Contribution The author wrote the initial draft of paper. The author also actively contributed in the algorithm proposal and the evaluations presented in the article. The final version of the article was created by the collaboration and inputs of all the authors.

4.6 Article PVI: "Finding tours for a set of interests"

Mohamed Abdel Maksoud, Gaurav Pandey (first co-author) and Shuaiqiang Wang. Finding tours for a set of interests. In Companion of The Web Conference 2018, pp. 215-218. International World Wide Web Conferences Steering Committee, 2018.

TABLE 2 Article Summarization

Article	Title	Field	Channel	Status
PI	Linear feature extraction for ranking	Document Ranking	Information Retrieval Journal 2018	published
PII	Vectors of pairwise item preferences	Recommender Systems	European Conference on Information Retrieval 2019	published
PIII	Recommending serendipitous items using transfer learning	Recommender Systems	Proceedings of the 27th ACM International Conference on Information and Knowledge Management 2018	published
PIV	Listwise recommendation approach with non-negative matrix factorization	Recommender Systems	International Conference on Intelligent Decision Technologies 2018	published
PV	CitySearcher: A city search engine for interests	Document Ranking	Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval 2017	published
PVI	Finding tours for a set of interests	Document Ranking	Companion of The Web Conference 2018	published

Aim The article addresses the problem of generating tours (or groups of cities) for a set of interests, by utilizing the given rankings of cities for interests.

Results We demonstrate the web-application Travición, that presents the user with the ranking of tours for a set of interests. It aims that each interest is satisfied by at least one city in a tour and the distance between the cities in the tour is not too large. It utilizes the *CitySearcher* algorithm from Article PV and utilizes the available ranking of cities for each interest. We present our novel techniques to calculate the ranking scores of tours and compare the results.

Author's Contribution The author wrote the initial draft of paper. The author also actively contributed in the algorithm proposal and the evaluations presented in the article. The final version of the paper was created by the collaboration and inputs of all the authors.

The presented articles are summarized in Table 2.

5 CONCLUSION

In this final chapter, we summarize the dissertation and present our limitations with future directions.

5.1 Summary

This dissertation answers five research questions and makes contributions in response to each research question, resulting in six published articles. The research work primarily focuses on the improvements in the ranking performances of two of the most widely used web applications: document ranking systems and recommender systems. The key aim while creating a ranking of results, is to order the results in such a way that the most useful result should come first, followed by other results in decreasing order of usefulness.

In our research, we found research gaps in the areas of document ranking and recommender systems. Thereafter, based on these gaps, we proposed algorithms to improve the ranking performances by creating new types of features and vectors as well as new algorithmic techniques.

We found that while there were efforts for feature selection for learning to rank, there were no efforts in the area of feature extraction. We presented *LifeRank*, that is a linear feature extraction algorithm to be used by learning to rank algorithms for document ranking. For this *LifeRank* creates a transformation matrix, that can be used to extract features from documents in lower dimension. Moreover, for recommender systems, we created an algorithm *Pref2Vec*, that creates vectors of pairwise item preferences. This enables the neural embedding process to use multiple levels of user ratings, that the existing algorithms were unable to utilize. The created vectors can then be used to create user vectors and item vectors, that can be used by recommender systems.

Also, we realized that the efforts in the field of serendipitous recommendations were limited, especially there are no existing efforts to use deep neural networks using serendipity scores. For this we proposed the *SerRec* algorithm

that utilizes the NCF framework by He et al. (2017) and uses transfer learning for recommending serendipitous items. Moreover, we proposed *LwRec* that presents a listwise approach to use ranking probabilities instead of ratings to improve recommendation performance.

Lastly, we demonstrated the utilization of *word2vec*, a famous neural embedding technique for the specific purpose of creating rankings of cities for user interests. For this we presented our algorithm *CitySearcher*. Moreover, this was also utilized to demonstrate the creation of ranking of tours (i.e. group of cities) for a set of interests.

5.2 Limitations and Future Directions

While our feature extraction algorithm *LifeRank* is a linear approach, it would be interesting to explore non-linear feature extraction approaches that could lead to further improvements in performance. Moreover, deep learning techniques could be applied for feature extraction. Secondly, *Pref2Vec* approach utilizes the preferences on items by the user, but it still ignores the magnitude of the preference. For example, it would consider a preference of a rating of 2 stars over 1 star, as the same as the preference of preference of 5 stars over 1 star. Therefore, in future we can develop algorithms that can include such magnitudes. Also, in the *LwRec* algorithm we used the ranking probability matrix for only top 1 item rankings, and it would be interesting to explore if creation of rankings with more items would lead to further improvements.

Moreover, we explore only the NCF framework for recommending serendipitous items, but for the same purpose we can design other deep neural networks and utilize using other available deep learning frameworks for this. Moreover, our recommendation algorithms do not consider the temporal aspect, i.e. the user preferences shown long time ago would be considered as important as the ones shown very recently. Therefore, it would be interesting to develop advanced recommendation algorithms in the future that would consider the time of the rating in order to improve recommendation performances. Moreover, all our recommendation techniques use only the ratings used by the users in order to make recommendations. We could also extend the algorithms so that they can include the metadata about the users and items for further improvements.

Also, we presented our *CitySearcher* and tour creation algorithms for the purpose of vertical search to satisfy a user's travel interests. However, the underlying methods are generic and can be used to other domains of document ranking. For example, *CitySearcher* algorithm can be used for any dataset to solve the problem of mismatched semantic relationships while searching for documents. Moreover, the tour search methods can be used to create a ranking of group of documents to satisfy multiple user queries. It would therefore be interesting to explore other applications of these algorithms.

Lastly, while we explore recommending serendipitous items, the most of

the work focuses on using relevance as the sole indicator for accessing the usefulness of items or documents. However, in future work it is important to also see how relevance and serendipity along with criteria like the novelty, fairness, transparency, etc., of the generated results can contribute to the usefulness of the results and their rankings. It would be interesting to create datasets specific to these criteria as well as to create advanced methods to bring improvements in rankings with respect to them.

YHTEENVETO (FINNISH SUMMARY)

Tehokkaiden ominaisuuksien, vektorien ja koneoppimisen hyödyntäminen sijoitustekniikoille

Dokumentteja hakevat järjestelmät ja suosittelujärjestelmät ovat kaksi käytetyimmistä nettisovelluksista. Edelliset hakevat dokumentteja, kun käyttäjä antaa hakutermin, jälkimmäiset ehdottavat tuotteita tai nimikkeitä käyttäjille sen mukaan, mitkä heidän mieltymyksensä olivat menneisyydessä. Tyypillisesti molemmat käyttävät järjestämistekniikoita ja niiden tuottamat tulokset esitetään luetteloiden muodossa. Päätavoite tulosten järjestämisessä on, että hyödyllisin tulos esiintyy luettelossa ensin ja muut tulokset seuraavat sitä laskevassa hyödyllisyysjärjestyksessä. Tässä väitöskirjassa esittelemme tutkimustyötä, jossa keskitytään parantamaan dokumentteja hakevien järjestelmien ja suosittelujärjestelmien tulosluetteloiden järjestämistekniikoita. Löysimme aukkoja tutkimuksessa näillä alueilla ja ehdotimme keinoja parantaa järjestämistoimintoja. Tämä tehtiin luomalla uudentyyppisiä dokumenttien ominaispiirteitä ja piirrevektoreita sekä uusia algoritmitekniikoita.

Ominaispiirrevalinnasta oppimista on tutkittu paljon, mutta ominaispiirteiden kokoamista ei ole tutkittu. Työssä esitellään *LifeRank*, joka on lineaarinen ominaispiirteiden kokoamista suorittava algoritmi. Järjestämään oppivat algoritmit voivat käyttää sitä dokumenttien sijoittamiseen tulosluetteloon. Lisäksi esitellään *Pref2Vec*- algoritmi, joka luo tuoteparipreferenssivektoreita neuraalista upotusta käyttäen. Tämä mahdollistaa useiden käyttäjäluokitustasojen käytön neuraalisessa upotuksessa, mitä nykyiset algoritmit eivät tue. Generoitujen vektoreiden avulla voidaan luoda käyttäjävektoreitä ja tuotevektoreita, joita suosittelujärjestelmät puolestaan voivat käyttää.

Havaitsimme myös, että onnekkailla suosituksilla on tehty vain vähän kokeiluja, varsinkaan sellaisia, joissa käytetään syväoppimista onnekassuosituksista. Tähän tarkoitukseen kehitimme *SerRec*- algoritmin, joka käyttää syväoppimista ja siirto-oppimista antaessaan onnekassuosituksia käyttäjille. Lisäksi työssä esitellään *LwRec*-algoritmi, joka käyttää suositusten parantamiseksi sijoitustodennäköisyyksiä tuloslistassa käyttäjäarviointien sijasta.

Työssä tutkitaan myös matkakohteiden suosittelua käyttäjän kohdeintresseihin perustuen. Tässä käytimme *word2vec*-teniikkaa, joka on neuraalinen upotustekniikka, sijoittamaan kaupunkeja matkakohdeintressien mukaan. Algoritmimme *CitySearcher* nojaa siihen. Lopuksi esittelemme myös edellistä algoritmia käyttävän retkenluomisalgoritmin, joka luo käyttäjän kohdeintressejä vastaavien kaupunkien kautta kulkevia retkisuosituksia.

Avainsanat: järjestäminen, tiedonhaku, suosittelujärjestelmä, syväoppiminen, neuraalinen upotus, onnekas suositus

REFERENCES

Abdel Maksoud, M., Pandey, G. & Wang, S. 2017. Citysearcher: A city search engine for interests. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, 1141–1144.

Abdel Maksoud, M., Pandey, G. & Wang, S. 2018. Finding tours for a set of interests. In Companion Proceedings of The Web Conference 2018. International World Wide Web Conferences Steering Committee. WWW '18, 215–218.

Adomavicius, G. & Tuzhilin, A. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge & Data Engineering 6, 734–749.

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G. et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In International conference on machine learning, 173–182.

André, P., Teevan, J. & Dumais, S. T. 2009. From x-rays to silly putty via uranus: Serendipity and its role in web search. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. New York, NY, USA: ACM. CHI '09, 2033–2036. doi:10.1145/1518701.1519009. ⟨URL:http://doi.acm.org/10.1145/1518701.1519009⟩.

Asgari, E. & Mofrad, M. R. 2015. Continuous distributed representation of biological sequences for deep proteomics and genomics. PloS one 10 (11), e0141287.

Baeza-Yates, R. & Ribeiro-Neto, B. 1999. Modern Information Retrieval. Addison Wesley.

Barkan, O. & Koenigstein, N. 2016. Item2vec: Neural item embedding for collaborative filtering. In MLSP, 1-6.

Bengio, Y., Ducharme, R., Vincent, P. & Janvin, C. 2003. A neural probabilistic language model. Journal of machine learning research 3, 1137–1155.

Bhingardive, S., Singh, D., V, R., Redkar, H. H. & Bhattacharyya, P. 2015. Unsupervised most frequent sense detection using word embeddings. In HLT-NAACL, 1238-1243.

Bordes, A., Usunier, N., Garcia-Durán, A., Weston, J. & Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In NIPS, 2787–2795.

Burges, C. J., Ragno, R. & Le, Q. V. 2007. Learning to rank with nonsmooth cost functions. In NIPS, 193–200.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N. & Hullender, G. 2005. Learning to rank using gradient descent. In ICML, 89–96.

Büttcher, S., Clarke, C. L. & Cormack, G. V. 2016. Information retrieval: Implementing and evaluating search engines. MIT Press.

Cao, Y., Xu, J., Liu, T.-Y., Li, H., Huang, Y. & Hon, H.-W. 2006. Adapting ranking SVM to document retrieval. In SIGIR, 186–193.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F. & Li, H. 2007. Learning to rank: from pairwise approach to listwise approach. In Proceedings of the 24th international conference on Machine learning. ACM, 129–136.

Chapelle, O., Chang, Y. & Liu, T.-Y. 2011. Future directions in learning to rank. Journal of Machine Learning Research 14, 91–100.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M. et al. 2016. Wide & deep learning for recommender systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems. ACM, 7–10.

Choi, S., Cha, S. & Tapper, C. C. 2010. A survey of binary similarity and distance measures. J. Systemics, Cybernetics and Informatics 8 (1), 43–48.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. & Kuksa, P. 2011. Natural language processing (almost) from scratch. Journal of Machine Learning Research 12 (Aug), 2493–2537.

Cossock, D. & Zhang, T. 2008. Statistical analysis of Bayes optimal subset ranking. IEEE Transactions on Information Theory 54 (11), 5140–5154.

Covington, P., Adams, J. & Sargin, E. 2016. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM conference on recommender systems. ACM, 191–198.

Crammer, K. & Singer, Y. 2001. Pranking with ranking. In NIPS, 641–647.

Deng, L., Hinton, G. & Kingsbury, B. 2013. New types of deep neural network learning for speech recognition and related applications: An overview. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 8599–8603.

De Campos, L. M., Fernández-Luna, J. M., Huete, J. F. & Rueda-Morales, M. A. 2010. Combining content-based and collaborative recommendations: A hybrid approach based on bayesian networks. International Journal of Approximate Reasoning 51 (7), 785–799.

Djuric, N., Wu, H., Radosavljevic, V., Grbovic, M. & Bhamidipati, N. 2015. Hierarchical neural language models for joint representation of streaming documents and their content. In WWW, 248–255.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E. & Darrell, T. 2014. Decaf: A deep convolutional activation feature for generic visual recognition. In International conference on machine learning, 647–655.

Ekstrand, M. D., Riedl, J. T. & Konstan, J. A. 2011. Collaborative filtering recommender systems. Foundations and Trends® in Human–Computer Interaction 4 (2), 81–173.

Freund, Y., Iyer, R., Schapire, R. E. & Singer, Y. 2003. An efficient boosting algorithm for combining preferences. Journal of Machine Learning Research 4 (1), 933–969.

Ganguly, D., Roy, D., Mitra, M. & Jones, G. J. 2015. Word embedding based generalized language model for information retrieval. In Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval. ACM, 795–798.

Geng, X., Liu, T., Qin, T. & Li, H. 2007. Feature selection for ranking. In SIGIR, 407–414.

Gopalan, P., Hofman, J. M. & Blei, D. M. 2015. Scalable Recommendation with Hierarchical Poisson Factorization. In UAI, 326–335.

Grbovic, M., Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V. & Sharp, D. 2015. E-commerce in your inbox: Product recommendations at scale. In SIGKDD, 1809–1818.

Grotov, A. & de Rijke, M. 2016. Online learning to rank for information retrieval: Sigir 2016 tutorial. In Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval. ACM, 1215–1218.

Grover, A. & Leskovec, J. 2016. Node2vec: Scalable feature learning for networks. In SIGKDD, 855–864.

Gunawardana, A. & Shani, G. 2009. A survey of accuracy evaluation metrics of recommendation tasks. Journal of Machine Learning Research 10, 2935–2962.

Gupta, P. & Rosso, P. 2012. Expected divergence based feature selection for learning to rank. In COLING, 431–440.

He, K., Zhang, X., Ren, S. & Sun, J. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, 770–778.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X. & Chua, T.-S. 2017. Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 173–182.

Herlocker, J., Konstan, J. A. & Riedl, J. 2002. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. Information Retrieval 5 (4), 287–310.

Hirschberg, J. & Manning, C. D. 2015. Advances in natural language processing. Science 349 (6245), 261–266.

Hofmann, K., Schuth, A., Whiteson, S. & de Rijke, M. 2013. Reusing historical interaction data for faster online learning to rank for ir. In Proceedings of the sixth ACM international conference on Web search and data mining. ACM, 183–192.

Hosmer Jr, D. W., Lemeshow, S. & Sturdivant, R. X. 2013. Applied Logistic Regression, Vol. 398. John Wiley & Sons. Wiley Series in Probability and Statistics.

Huang, S., Wang, S., Liu, T.-Y., Ma, J., Chen, Z. & Veijalainen, J. 2015. Listwise collaborative filtering. In SIGIR, 343–352.

Järvelin, K. & Kekäläinen, J. 2002. Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems 20 (4), 422–446.

Joachims, T., Finley, T. & Yu, C.-N. J. 2009. Cutting-plane training of structural SVMs. Machine Learning 77 (1), 27–59.

Joachims, T., Li, H., Liu, T.-Y. & Zhai, C. 2007. Learning to rank for information retrieval (LR4IR 2007). ACM SIGIR Forum 41 (2), 58–62.

Joachims, T., Swaminathan, A. & de Rijke, M. 2018. Deep learning with logged bandit feedback. In International Conference on Learning Representations.

Kabbur, S., Ning, X. & Karypis, G. 2013. FISM: Factored item similarity models for top-N recommender systems. In SIGKDD, 659–667.

Kendall, M. G. 1948. Rank Correlation Methods. C. Griffin.

Koren, Y., Bell, R. & Volinsky, C. 2009. Matrix factorization techniques for recommender systems. Computer 42 (8), 30–37.

Koren, Y. & Bell, R. 2015. Advances in collaborative filtering. In Recommender systems handbook. Springer, 77–118.

Kotkov, D., Konstan, J. A., Zhao, Q. & Veijalainen, J. 2018. Investigating serendipity in recommender systems based on real user feedback. In Proceedings of SAC 2018: Symposium on Applied Computing. ACM.

Kotkov, D., Veijalainen, J. & Wang, S. 2019. How does serendipity affect diversity in recommender systems? A serendipity-oriented greedy algorithm. Computing, 19 pages. (published online in 2018).

Kotkov, D., Wang, S. & Veijalainen, J. 2016. A survey of serendipity in recommender systems. Knowledge-Based Systems 111, 180–192.

Kusner, M., Sun, Y., Kolkin, N. & Weinberger, K. 2015. From word embeddings to document distances. In International Conference on Machine Learning, 957–966.

Lai, H., Pan, Y., Tang, Y. & Yu, R. 2013. FSMRank: Feature selection algorithm for learning to rank. IEEE Transactions on Neural Networks and Learning Systems 24 (6), 940–952.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K. & Dyer, C. 2016. Neural architectures for named entity recognition. In HLT-NAACL, 260-270.

Laporte, L., Flamary, R., Canu, S., Déjean, S. & Mothe, J. 2014. Nonconvex regularizations for feature selection in ranking with sparse SVM. IEEE Transactions on Neural Networks and Learning Systems 25 (6), 1118–1130.

Lawson, C. & Hanson, R. 1995. Solving Least Square Problems, Vol. 15. SIAM. Classics in Applied Mathematics.

Le, Q. & Mikolov, T. 2014. Distributed representations of sentences and documents. In ICML, 1188-1196.

Lee, D. D. & Seung, H. S. 2000. Algorithms for Non-negative Matrix Factorization. In NIPS, 556–562.

Lee, J., Kim, S., Lebanon, G. & Singer, Y. 2013. Local Low-rank Matrix Approximation. In ICML, II-82–II-90.

Li, P., Wu, Q. & Burges, C. J. 2007. McRank: Learning to rank using multiple classification and gradient boosting. In NIPS, 897–904.

Linden, G., Smith, B. & York, J. 2003. Amazon.com Recommendations: Item-to-item Collaborative Filtering. IEEE Internet Computing 7 (1), 76–80.

Liu, N. N. & Yang, Q. 2008. EigenRank: A Ranking-oriented Approach to Collaborative Filtering. In SIGIR, 83–90.

Liu, T.-Y. 2009. Learning to rank for information retrieval. Foundations and Trends® in Information Retrieval 3 (3), 225–331.

Liu, T.-Y. 2011. Learning to Rank for Information Retrieval. Springer Science & Business Media.

Lops, P., de Gemmis, M. & Semeraro, G. 2011. Content-based recommender systems: State of the art and trends. In Recommender Systems Handbook. Springer, 73–105.

Lu, Q., Chen, T., Zhang, W., Yang, D. & Yu, Y. 2012. Serendipitous personalized ranking for top-n recommendation. In 2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, Vol. 1. IEEE, 258–265.

McNee, S. M., Riedl, J. & Konstan, J. A. 2006. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In CHI'06 extended abstracts on Human factors in computing systems. ACM, 1097–1101.

Metzler, D. A. 2007. Automatic feature selection in the markov random field model for information retrieval. In CIKM. ACM, 253–262.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. 2013a. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. & Dean, J. 2013b. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, 3111–3119.

Murphy, K. P. 2012. Machine Learning: A Probabilistic Perspective. The MIT Press.

Naini, K. D. & Altingövde, I. S. 2014. Exploiting result diversification methods for feature selection in learning to rank. In ECIR. Springer, 455–461.

Nalisnick, E., Mitra, B., Craswell, N. & Caruana, R. 2016. Improving document ranking with dual word embeddings. In Proceedings of the 25th International Conference Companion on World Wide Web. International World Wide Web Conferences Steering Committee, 83–84.

Ng, A. Y. 2004. Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In Proceedings of the 21st International Conference on Machine Learning. ACM, 8 pp.

Ng, P. 2017. dna2vec: Consistent vector representations of variable-length k-mers. arXiv preprint arXiv:1701.06279.

Nguyen, T. T., Hui, P.-M., Harper, F. M., Terveen, L. & Konstan, J. A. 2014. Exploring the filter bubble: the effect of using recommender systems on content diversity. In Proceedings of the 23rd international conference on World wide web. ACM, 677–686.

Ning, X. & Karypis, G. 2011. SLIM: Sparse Linear Methods for Top-N Recommender Systems. In ICDM, 497–506.

Niu, S., Guo, J., Lan, Y. & Cheng, X. 2012. Top-k learning to rank: labeling, ranking and evaluation. In Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval. ACM, 751–760.

Pan, F., Converse, T., Ahn, D., Salvetti, F. & Donato, G. 2009. Feature selection for ranking using boosted trees. In CIKM. ACM, 2025–2028.

Pan, W. & Chen, L. 2013. GBPR: Group Preference Based Bayesian Personalized Ranking for One-class Collaborative Filtering. In IJCAI, 2691–2697.

Pandey, G., Kotkov, D. & Semenov, A. 2018a. Recommending serendipitous items using transfer learning. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management. ACM, 1771–1774.

Pandey, G., Ren, Z., Wang, S., Veijalainen, J. & de Rijke, M. 2018b. Linear feature extraction for ranking. Information Retrieval Journal 21 (6), 481–506.

Pandey, G., Wang, S., Ren, Z. & Chang, Y. 2019. Vectors of pairwise item preferences. In European Conference on Information Retrieval. Springer, 323–336.

Pandey, G. & Wang, S. 2018. Listwise recommendation approach with non-negative matrix factorization. In Intelligent Decision Technologies 2018, Proceedings of the 10th International Conference on Intelligent Decision Technologies (KES-IDT 2018). Springer, 22–32.

Pazzani, M. J. & Billsus, D. 2007. Content-based recommendation systems. In The adaptive web. Springer, 325–341.

Peffers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S. 2007. A design science research methodology for information systems research. Journal of management information systems 24 (3), 45–77.

Perozzi, B., Al-Rfou, R. & Skiena, S. 2014. Deepwalk: Online learning of social representations. In SIGKDD, 701–710.

Qin, T., Liu, T.-Y., Xu, J. & Li, H. 2010. LETOR: A benchmark collection for research on learning to rank for information retrieval. Information Retrieval 13 (4), 346–374.

Qin, T. & Liu, T.-Y. 2013. Introducing LETOR 4.0 Datasets. arXiv 1306.2597.

Ranjan, R., Patel, V. M. & Chellappa, R. 2019. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 41 (1), 121–135.

Rendle, S., Freudenthaler, C., Gantner, Z. & Schmidt-Thieme, L. 2009. BPR: Bayesian personalized ranking from implicit feedback. In UAI, 452–461.

Rendle, S. 2010. Factorization machines. In ICDM, 995–1000.

Rendle, S. 2012. Factorization machines with libFM. ACM Transactions on Intelligent Systems and Technology 3 (3), 57:1–57:22.

Salakhutdinov, R. & Mnih, A. 2007. Probabilistic matrix factorization. In NIPS, 1257–1264.

Schuth, A., Oosterhuis, H., Whiteson, S. & de Rijke, M. 2016. Multileave gradient descent for fast online learning to rank. In Proceedings of the Ninth ACM International Conference on Web Search and Data Mining. ACM, 457–466.

Schwenk, H. 2007. Continuous space language models. Computer Speech Lang. 21 (3), 492–518.

Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Oliver, N. & Hanjalic, A. 2012. CLiMF: Learning to Maximize Reciprocal Rank with Collaborative Less-is-more Filtering. In RecSys, 139–146.

Socher, R., Chen, D., Manning, C. D. & Ng, A. 2013. Reasoning with neural tensor networks for knowledge base completion. In NIPS, 926–934.

Socher, R., Lin, C. C., Manning, C. & Ng, A. Y. 2011. Parsing natural scenes and natural language with recursive neural networks. In Proceedings of the 28th international conference on machine learning (ICML-11), 129–136.

Tsai, M.-F., Liu, T.-Y., Qin, T., Chen, H.-H. & Ma, W.-Y. 2007. FRank: A ranking method with fidelity loss. In SIGIR. ACM, 383–390.

Turian, J., Ratinov, L. & Bengio, Y. 2010. Word representations: a simple and general method for semi-supervised learning. In Proceedings of the 48th annual meeting of the association for computational linguistics. Association for Computational Linguistics, 384–394.

Turney, P. D. 2013. Distributional semantics beyond words: Supervised learning of analogy and paraphrase. TACL 1, 353-366.

Valizadegan, H., Jin, R., Zhang, R. & Mao, J. 2009. Learning to rank by optimizing NDCG measure. In NIPS, 1883–1891.

Volkovs, M. & Zemel, R. S. 2009. Boltzrank: Learning to maximize expected ranking gain. In ICML, 1089–1096.

Wang, S., Sun, J., Gao, B. J. & Ma, J. 2014. VSRank: A novel framework for ranking-based collaborative filtering. ACM Transactions on Intelligent Systems and Technology 5 (3), 51:1–51:24.

Weimer, M., Karatzoglou, A., Le, Q. V. & Smola, A. 2007. CofiRank: Maximum margin matrix factorization for collaborative ranking. In NIPS, 1593–1600.

Wu, Y., DuBois, C., Zheng, A. X. & Ester, M. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In Proceedings of the Ninth ACM International Conference on Web Search and Data Mining. ACM, 153–162.

Xu, J. & Li, H. 2007. Adarank: a boosting algorithm for information retrieval. In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 391–398.

Yamaba, H., Tanoue, M., Takatsuka, K., Okazaki, N. & Tomita, S. 2013. On a serendipity-oriented recommender system based on folksonomy and its evaluation. Procedia Computer Science 22, 276–284.

Ye, X., Shen, H., Ma, X., Bunescu, R. & Liu, C. 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In Proceedings of the 38th international conference on software engineering. ACM, 404–415.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L. & Leskovec, J. 2018. Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, 974–983.

Young, T., Hazarika, D., Poria, S. & Cambria, E. 2018. Recent trends in deep learning based natural language processing. IEEE Computational Intelligence Magazine 13 (3), 55–75.

Yu, H., Oh, J. & Han, W. 2009. Efficient feature weighting methods for ranking. In CIKM. ACM, 1157–1166.

Yue, Y., Finley, T., Radlinski, F. & Joachims, T. 2007. A support vector method for optimizing average precision. In SIGIR. ACM, 271–278.

Zhang, S., Yao, L., Sun, A. & Tay, Y. 2019. Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys (CSUR) 52 (1), 5.

Zhang, Y., Pezeshki, M., Brakel, P., Zhang, S., Bengio, C. L. Y. & Courville, A. 2017. Towards end-to-end speech recognition with deep convolutional neural networks. arXiv preprint arXiv:1701.02720.

Zhang, Y. C., Séaghdha, D. Ó., Quercia, D. & Jambor, T. 2012. Auralist: introducing serendipity into music recommendation. In Proceedings of the fifth ACM international conference on Web search and data mining. ACM, 13–22.

Zheng, Q., Chan, C.-K. & Ip, H. H. 2015. An unexpectedness-augmented utility model for making serendipitous recommendation. In Industrial Conference on Data Mining. Springer, 216–230.

Zoghi, M., Tunys, T., Ghavamzadeh, M., Kveton, B., Szepesvari, C. & Wen, Z. 2017. Online learning to rank in stochastic click models. In Proceedings of the 34th International Conference on Machine Learning, Vol. 70. JMLR. org. PMLR, 4199–4208.

Zou, W. Y., Socher, R., Cer, D. M. & Manning, C. D. 2013. Bilingual word embeddings for phrase-based machine translation. In EMNLP, 1393-1398.

ORIGINAL PAPERS

PI

LINEAR FEATURE EXTRACTION FOR RANKING

by

Gaurav Pandey, Zhaochun Ren, Shuaiqiang Wang, Jari Veijalainen, Maarten de Rijke 2018

Information Retrieval Journal

https://doi.org/10.1007/s10791-018-9330-5

Noname manuscript No.
(will be inserted by the editor)

Linear Feature Extraction for Ranking

Gaurav Pandey · Zhaochun Ren · Shuaiqiang Wang · Jari Veijalainen · Maarten de Rijke

Received: date / Accepted: date

Abstract We address the feature extraction problem for document ranking in information retrieval. We then propose LifeRank, a Linear feature extraction algorithm for Ranking. In LifeRank, we regard each document collection for ranking as a matrix, referred to as the *original matrix*. We optimize a *transformation matrix*, so that a new matrix (dataset) can be generated as the product of the original matrix and a transformation matrix. The transformation matrix projects high-dimensional document vectors into lower dimensions. Theoretically, there could be very large transformation matrices, each leading to a new generated matrix. In LifeRank, we produce a transformation matrix so that the generated new matrix can match the learning to rank problem. Extensive experiments on benchmark datasets show the performance gains of LifeRank in comparison with state-of-the-art feature selection algorithms.

Keywords Feature extraction · Dimension reduction · Learning to rank · Information retrieval

Gaurav Pandey
University of Jyväskylä
E-mail: gaurav.g.pandey@student.jyu.fi

Zhaochun Ren
JD.com
E-mail: renzhaochun@jd.com

Shuaiqiang Wang
University of Jyväskylä
E-mail: shuaiqiang.wang@jyu.fi

Jari Veijalainen
University of Jyväskylä
E-mail: jari.veijalainen@jyu.fi

Maarten de Rijke
University of Amsterdam
E-mail: derijke@uva.nl

1 Introduction

Document ranking is an essential component of information retrieval systems and web search engines. Recently, machine learning-based ranking techniques, referred to as "learning to rank," have given rise to an active and growing research area, both in the information retrieval and machine learning communities [10, 15, 21, 39, 59]. A large number of learning to rank algorithms have been proposed, which incorporate more and more useful features, aiming to improve the performance of the ranking algorithms [33]. In a supervised setting, they first collect a set of training data, which includes a set of queries, each associated with a list of documents labeled by relevance degrees; with the training dataset, they train a ranking model that can order unseen documents according to their degree of relevance [20]. In this situation, dimension reduction inevitably becomes an important issue [16].

Firstly, dimension reduction can enhance the accuracy for many machine learning problems, including learning to rank. With dimension reduction techniques, a small set of more discriminative and less redundant features can be selected or generated for learning. Thus, better results could be achieved, as overfitting becomes less likely [38]. Also, the generalization ability of machine learning models could depend on the radius of training data points, which may decrease when the number of features decreases [5, 16, 56, 57].

Secondly, large number of features leads to high complexity in most learning to rank algorithms. Therefore, dimension reduction often leads to significant improvements in training and prediction efficiency, while maintaining, or having a limited negative impact on, accuracy. With accuracy being the primary metric, efficiency has also emerged as a crucial issue for evaluating learning to rank algorithms [10, 11, 55]. Training datasets and ranking features continue to expand, so as to obtain more accurate models. Furthermore, as a consequence of the dynamic character of the Web, ranking models need to be re-learned repeatedly, and the interval between re-learning procedures decreases sharply [33]. With dimension reduction techniques, fewer features are used, resulting in more efficient training and prediction.

Generally, there are two types of dimension reduction algorithms: feature *selection* and feature *extraction*. The former aims to select a subset of the original features for learning, while the latter attempts to generate a small set of new features from the original features [5, 35, 58]. Recently, feature selection for ranking has been investigated intensively [16, 17, 26, 29, 37, 40, 60]. To the best of our knowledge, the advantages of feature *extraction* have not yet been explored in learning to rank.

In this study, we address the *feature extraction problem for learning to rank*. In comparison with feature selection, the feature extraction problem has a much larger search space. For example, given n original features, feature selection selects a subset of features of size k (where $k < n$) for learning. Here, for a particular value of k, the search space of the problem contains $\binom{n}{k}$ possible solutions. The full search space that can include any number of features (i.e., all values of k in range 1 to n), would lead to $2^n - 1$ solutions. In comparison, for linear feature extraction, each extracted feature is a linear combination of original n features. Since the coefficient associated with each original feature can be any real number, the search space becomes infinite. The search space of non-linear feature extraction would be even larger, as it also includes

solutions involving non-linear combinations of features (e.g. polynomial combinations). Hence, with a larger search space, feature extraction has a greater possibility to achieve better performance than feature selection.

To address the problem of linear feature extraction for learning to rank, we propose LifeRank, a Linear feature extraction algorithm for Ranking. LifeRank regards each dataset for training, validation or test as a matrix, referred to as an *original matrix*, where each row vector represents a document with a set of features. With a given original matrix for training $\mathbf{X}$, LifeRank attempts to discover a transformation matrix $\mathbf{T}$, so that a new matrix (dataset) $\mathbf{X}'$ can be generated as the product of the original matrix and a transformation matrix, i.e., $\mathbf{X}' = \mathbf{X}\mathbf{T}$. Thus $\mathbf{T}$ projects high-dimensional document vectors in $\mathbf{X}$ into lower-dimensional ones in $\mathbf{X}'$. Theoretically, there could be a very large number of possible transformation matrices, each leading to a new generated matrix. LifeRank attempts to discover a transformation matrix to transform the original matrix (dataset) into a low-rank one for dimension reduction, on which learning to rank algorithms can achieve optimum results in comparison with other dimension-reduced matrices.

Our problem formulation is similar to principal component analysis (PCA) [23], and thus our algorithm LifeRank can be understood from the perspective of PCA. PCA is one of the most popular dimension reduction techniques in machine learning. When PCA is performed using singular valued decomposition (SVD) [28], the given matrix $\mathbf{X}$ can be approximately decomposed into three low-rank matrices $\mathbf{X} \approx \mathbf{P}\mathbf{\Sigma}\mathbf{Q}^\top$. Here, $\mathbf{\Sigma}$ is composed of the singular values of $\mathbf{X}$, $\mathbf{P}$ and $\mathbf{Q}$ are composed of the left and right singular vectors of $\mathbf{X}$ respectively, and $\mathbf{P}^\top\mathbf{P} = \mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$ is equal to the identity matrix. Thus a new matrix $\mathbf{X}' = \mathbf{P}\mathbf{\Sigma} \approx \mathbf{X}\mathbf{Q}$. However, it should be noted that while PCA calculates $\mathbf{X}'$ as an approximation of $\mathbf{X}$, in LifeRank $\mathbf{X}$ is transformed to $\mathbf{X}'$ using a transformation matrix.

In LifeRank, we formulate the learning to rank task by using a classical pairwise loss function. A pairwise loss function is used because such functions are fundamental, straightforward and intuitive for ranking. Besides, pairwise loss functions are consistent with the assumption that the labels of documents to rank lie in a rank-differentiable probability space [27], and they are upper bounds of measure-based ranking errors [12]. In the generated matrix, the column vectors represent the features. Since optimization over orthogonal features is beneficial to many machine learning problems [48, 49], we utilize the Lagrange multipliers method [1, 4] to impose orthonormality constraints on the column (feature) vectors of the transformed matrix, and then use gradient descent for optimization. With the transformation matrix $\mathbf{T}$, the training, validation and test datasets can be directly generated with matrix product.

Note that (1) LifeRank generalizes feature selection algorithms for the learning to rank task. Feature selection can be regarded as optimizing a transformation matrix $\mathbf{T}$ so that the column vectors of $\mathbf{T}$ meet the orthonormality constraints and each element in $\mathbf{T}$ can only be either 0 or 1. (2) Although some deep learning-based ranking algorithms [47] also aim to generate a set of features for ranking, our problem is completely different: we try to construct our features based on some predesigned ranking features like term frequency (TF) and inverse document frequency (IDF), which have been comprehensively used in conventional learning to rank algorithms like Ranking SVM [9, 21] and RankBoost [15]. Deep learning-based algorithms, however, try to

build features based on word-level features in a corpus that differ substantially from conventional ranking features.

Our main contributions are as follows: (1) We address the feature extraction problem for learning to rank. Feature extraction is a category of comprehensively used dimension reduction techniques in many machine learning problems for performance gains in accuracy and efficiency, but to the best of our knowledge, feature extraction and its advantages have not been explored in learning to rank yet. (2) We propose LifeRank, a linear feature extraction algorithm that generates datasets to be utilized by the learning to rank task. (3) We perform extensive experiments on benchmark datasets and present the performance gains of LifeRank in comparison with the state-of-the-art feature selection algorithms.

The remainder of the paper is organized as follows. Section 2 reviews related work; Section 3 defines the feature extraction problem for ranking; Section 4 proposes LifeRank, a gradient descent-based algorithm. Section 5 introduces our experimental setup. Section 6 reports the experimental results, and Section 7 concludes the paper.

2 Related work

We discuss three types of related work: learning to rank, feature selection for ranking, and feature extraction for ranking.

2.1 Learning to rank for information retrieval

Learning to rank has received increased attention from both the machine learning and information retrieval community. While there is growing interest in *online* learning to rank [46] and in *counterfactual* learning to rank from online data [22], the bulk of the work on learning to rank concerns *offline* learning to rank, where explicit human annotations are used to label query, document pairs. Offline learning to rank is the focus of this paper. Given its effectiveness, many algorithms have been proposed, which mainly fall into three categories [11, 32]: pointwise, pairwise, and listwise.

Pointwise approaches, such as Pranking [14], McRank [31] and Subset Ranking [13], view each document in the training dataset as a learning instance, and utilize a classification or regression technique to predict the relevance categories or numerical/ordinal relevance scores for unlabeled data. Pairwise approaches, such as Ranking SVM [9, 21], RankBoost [15], RankNet [6], FRank [52], LambdaRank [7], and BoltzRank [54], regard a pair of documents as a learning instance, and try to learn a binary classifier that can predict the more relevant document to the given query from each pair of documents. Then the ranked lists of documents can be aggregated based on the pairwise preferences of the documents. Listwise approaches, such as ListNet [10], SVM-MAP [61], NDCGBoost [53], take the entire ranked list of documents as a learning instance, and attempt to construct a ranking model that can directly predict the full rankings of the documents. Recently, some hybrid algorithms have been proposed, such as FocusedRank [39], MixRank [8], targeting improvements in learning accuracy, efficiency, or both. More algorithms are surveyed in [11, 32, 33].

With the incorporation of more and more useful features for performance gains, dimension reduction inevitably becomes an important issue in the ranking problem [16]. With effective dimension reduction techniques, not only the efficiency of the algorithms could be improved, but also accuracy could be enhanced as a result of using more discriminative features with less redundancy and noise. Furthermore, the generalization of the ranking model can also be increased as a result of using fewer features [16].

2.2 Feature selection for ranking

Recently, considerable efforts have been made on feature selection for ranking. Geng et al [16] present GAS, one of the first attempts to incorporate the importance and similarity of features for ranking. In particular, it evaluates the importance of features with ranking metrics like MAP [3] and NDCG [19], and estimates the similarity between features using agreement between rankings, e.g., with Kendall τ correlation coefficient [24]. Then it greedily selects a subset of features with maximum total importance scores and minimum total similarity scores. Metzler [34] proposes a greedy feature selection algorithm to be used within the Markov random field model for information retrieval. The model automatically generates models that are more effective than, or as effective as, models created by carefully selecting the features manually. Pan et al [40] investigate a boosted regression trees-based feature selection algorithm. It evaluates the importance of the features based on boosted trees. Then it selects features by maximizing the discounted importance of the features, where the importance of each feature is discounted by feature similarity. Yu et al [60] propose RankWrapper and RankSelect, two feature weighting and selection algorithms for learning to rank. They utilize ranking distances of nearest data points in order to identify the key features for ranking, demonstrating significant efficiency gains in comparison with GAS.

Gupta and Rosso [17] present a Kullback-Leibler (KL) divergence-based divergence metric, and select a subset of features for ranking based on features' expected divergence over the relevance classes and the importance of features. Lai et al [26] propose a joint convex optimization formulation for minimizing ranking errors while simultaneously conducting feature selection. This optimization formulation provides a flexible framework in which various importance measures and similarity measures of the features can easily be incorporated. Naini and Altingövde [37] adopt three greedy diversification strategies, maximal marginal relevance, MaxSum dispersion and modern portfolio theory, to the problem of feature selection for ranking. Laporte et al [29] propose a general framework for feature selection in learning to rank based on support vector machine (SVM); they investigate both classical convex regularizations (such as $L1$ and weighted $L1$) and non-convex regularization terms (such as log penalty, Minimax Concave Penalty (MCP) and Lp pseudo norm with $p < 1$). Furthermore, they provided an accelerated proximal approach for solving the convex problems and a re-weighted $L1$ scheme to address the non-convex regularizations.

All of these algorithms are meant to address *feature selection* for ranking. To the best of our knowledge, there is no work targeting *feature extraction* for ranking.

2.3 Feature extraction techniques

Feature extraction has been used extensively used in various machine learning scenarios for performance gains in terms of accuracy and efficiency. Given its effectiveness, many approaches have been proposed, which are either linear or non-linear algorithms.

The main linear technique for feature extraction is principal component analysis (PCA) [23], which performs a linear mapping of high-dimensional data into a lower-dimensional space in such a way that the variance of the data in the low-dimensional representation is maximized. Canonical-correlation analysis (CCA) [18] is another popular linear feature extraction algorithm, which attempts to discover linear combinations of the original features that have maximal correlation with each other. In addition, several probabilistic algorithms, including probabilistic PCA [51], probabilistic CCA [2] and probabilistic partial CCA [36], have been proposed, where a set of latent variables are introduced for probabilistically interpreting these models.

Non-linear feature extraction algorithms can combine the original features to generate a set of features in a non-linear way. For example, the locally linear embedding (LLE) method [44] learns the global structure of non-linear manifolds to yield low-dimensional, neighborhood-preserving embeddings of high-dimensional inputs. Isomap [50] is capable of discovering the non-linear degrees of freedom that underly complex natural observations. It can efficiently compute a globally optimal solution and can be guaranteed to converge asymptotically to the true structure. Besides, some kernel techniques have been proposed to transform linear feature extraction algorithms into nonlinear ones. For example, kernel PCA [45] is a non-linear form of principal component analysis (PCA), which can efficiently compute principal components in high-dimensional feature spaces through the use of integral operator kernel functions.

Although feature extraction techniques have been extensively investigated and shown to demonstrate promising performance gains, to the best of our knowledge, they have not been explored yet in the context of the ranking problem.

3 Problem statement

3.1 Learning to rank for information retrieval

Let $\mathcal{X}$ be a collection of documents, each represented by a vector of feature values. In information retrieval systems, given a query q, a list of documents from $\mathcal{X}$ is returned as search results, where the documents are ranked according to their estimated relevance to q. Given a query q, the ground truth, i.e., relevance judgments of documents with respect to q (produced by human experts) is defined as a function $rel : \mathcal{X} \rightarrow \mathbb{N}_0$, where $\mathbb{N}_0$ is the set of natural numbers (including 0).

Let $f : \mathcal{X} \rightarrow \mathbb{R}$ be a ranking function assigning real valued relevance scores to documents. The goodness of ranking functions can be evaluated by a measure s, such as precision at n ($P@n$), mean average precision (MAP) [3], or normalized discounted cumulative gain ($NDCG@n$) [19].

Definition 1 (Learning to rank) *Given a training dataset $\mathcal{X}$ and an evaluation measure s, the problem of learning to rank is to learn a ranking function f from $\mathcal{X}$ such that $s(f)$ is maximized.*

3.2 Dimension reduction for ranking

In learning to rank, each dataset $\mathcal{X}$ can be regarded as a document matrix $\mathbf{X}_{m\times n}$ with m rows (documents) and n columns (features). In particular, $\mathbf{x}_i$ is the i-th row of $\mathbf{X}$, and ${\mathbf{x}_i}^\top$ is a n-dimensional (column) vector that represents a document with n features. Let $g : \mathbb{R}^n \to \mathbb{R}^k$ ($k \leq n$) be a mapping that projects an n-dimensional vector space into a k-dimensional space. Let $L(\cdot)$ be the loss function for the learning to rank task. Our problem is to discover a mapping function g such that the obtained dataset $\mathbf{X}' = g(\mathbf{X})$ minimizes the loss function.

Definition 2 (Dimension reduction for ranking) *Let $\mathbf{X}_{m\times n}$ be a document matrix with m columns and n rows, where each column ${\mathbf{x}_i}^\top$ is a n-dimensional vector, representing a document with n features. Let $\mathcal{G}$ be the set of all possible mapping functions, where each element $g : \mathbb{R}^n \to \mathbb{R}^k$ ($k \leq n$) is used to project an n-dimensional vector space into a k-dimensional space. The dimension reduction for the learning to rank task tries to discover an optimum mapping function $g^* \in \mathcal{G}$ such that:*

$$\arg\min_{g\in\mathcal{G}} L(g(\mathbf{X})), \tag{1}$$

where $L(\cdot)$ is the loss function for the learning to rank task. Then the new dataset can be generated with $g^*(\mathbf{X})$.

In this paper, we consider linear feature extraction for learning to rank as it is the simplest and most straightforward feature extraction technique in machine learning. Here, each generated feature is a linear combination of the original features. It utilizes a transformation matrix $\mathbf{T}$ to achieve the effectiveness of the mapping function, aiming to discover an optimal matrix $\mathbf{T}$ such that the obtained dataset $\mathbf{X}' = \mathbf{XT}$ results in a minimal value of the loss function.

The problem can be understood from the perspective of PCA [23]. Using PCA, the given matrix $\mathbf{X}$ can be approximately decomposed into three lower-rank matrices:

$$\mathbf{X} \approx \mathbf{P\Sigma Q}^\top, \tag{2}$$

where $\mathbf{\Sigma}$ is composed of the singular values of $\mathbf{X}$, $\mathbf{P}$ and $\mathbf{Q}$ are composed of the left and right singular vectors of $\mathbf{X}$ respectively, and $\mathbf{P}^\top\mathbf{P} = \mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$ (the identity matrix). Thus, a new matrix $\mathbf{X}'$ can be generated as follows:

$$\mathbf{X}' = \mathbf{P\Sigma} \approx \mathbf{XQ}. \tag{3}$$

The role of the transformation matrix $\mathbf{T}$ in LifeRank is very similar to the right singular matrix $\mathbf{Q}$ in PCA, where $\mathbf{Q}$ maps the document vectors to another space spanned by the columns of $\mathbf{Q}$ before transforming them through $\mathbf{\Sigma}$ and going back through $\mathbf{P}$. Hence, in LifeRank we consider the orthonormality constraints of $\mathbf{T}$ in our optimization process, i.e., $\mathbf{T}^\top\mathbf{T} = \mathbf{I}$.

Definition 3 (Constrained linear feature extraction for ranking) *Let $\mathbf{X}_{m\times n}$ be a document matrix, where the transpose of each row, i.e., ${\mathbf{x}_i}^\top = \mathbf{d}_i$ is a n-dimensional vector, representing a document with n features. Linear feature extraction for ranking aims to optimize a transformation matrix $\mathbf{T}_{n\times k}$ by solving the following optimization problem, so that a new document matrix $\mathbf{X}'_{m\times k} = \mathbf{X}_{m\times n}\mathbf{T}_{n\times k}$ can be generated, where each document vector $\mathbf{d}_i$ can be projected into k-dimensional vector $\mathbf{d}'_i = \mathbf{T}^\top\mathbf{d}_i$:*

$$\arg\min_{\mathbf{T}} \quad L(\mathbf{XT}) \text{ such that } \mathbf{T}^\top\mathbf{T} = \mathbf{I}, \tag{4}$$

where $L(\cdot)$ is the loss function for the learning to rank task.

Based on the optimized mapping function g, the new dataset can be generated by taking the product of the original matrix and the transformation matrix, i.e., $\mathbf{X}' = \mathbf{XT}$.

We have used the example of PCA to help us explain the mechanism of LifeRank. However, it should be noted that in PCA $\mathbf{X}'$ is calculated as an approximation of $\mathbf{X}$, whereas in LifeRank we generate a transformed representation of the initial matrix, in order to achieve a better ranking performance. Hence, unlike PCA, $\mathbf{X}'$ as computed in Definition 3 is not an approximation of $\mathbf{X}$, but a transformation.

4 The LifeRank algorithm

Given a high-dimensional dataset $\mathcal{X}$, LifeRank generates a new low-dimensional dataset $\mathcal{X}'$ in two phases. In the first phase, LifeRank first preprocesses the training dataset $\mathcal{X}$ into an original matrix $\mathbf{X}$. Then LifeRank optimizes the transformation matrix $\mathbf{T}$ for $\mathbf{X}$ according to the loss function in Equation 4. In the second phase, LifeRank generates low-dimensional training, validation and test matrices with the projection of $\mathbf{T}$. Then LifeRank constructs new datasets based on the low-dimensional data matrices.

4.1 Phase I: Generation of the transformation matrix

In this study, we utilize a classic pairwise learning to rank loss function to implement the function $L(\cdot)$ in Definition 3. Pairwise loss functions are chosen because apart from being relatively simple and straightforward, they are also intuitive choices for ranking. Besides, with the assumption that the labels of documents to rank lie in a rank-differentiable probability space, pairwise loss functions are consistent [27] and provide upper bounds for measure-based ranking errors like NDCG [12]. Thus, minimizing a pairwise loss function will maximize the ranking measures [27].

First of all, the training dataset $\mathcal{X}$ is preprocessed into an original matrix $\mathbf{X}$ and other information $\mathbf{I}_X$ consisting of identities of the documents and queries, relevance labels, etc. Let $D = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_m\}$ be the set of columns (document vectors) in the matrix $\mathbf{X}^\top_{m\times n}$. We regard each pair of vectors $(\mathbf{d}_i, \mathbf{d}_j) \in D \times D$ as an instance, and the label $y_{i,j} \in \{+1, -1\}$ indicates whether the relevance of the i-th document is higher or lower than the j-th document, corresponding to the given query. Let

$\{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_k\}$ be the column vectors of $\mathbf{T}$. We try to discover a k-dimensional vector of weights $\mathbf{w}$ such that:

$$\begin{aligned} &\arg\min_{\mathbf{T},\mathbf{w},b} \sum_{\forall(\mathbf{d}_i,\mathbf{d}_j),i\neq j} \log\left(1+e^{-y_{i,j}\left(\mathbf{w}^\top\mathbf{T}^\top(\mathbf{d}_i-\mathbf{d}_j)+b\right)}\right)+\frac{\lambda}{2}||\mathbf{w}||^2 \\ &\text{such that } \mathbf{t}_i^\top\mathbf{t}_j=\begin{cases}1, & i=j\\ 0, & i\neq j\end{cases}, \forall i,j=1,2,\ldots,k, \end{aligned} \tag{5}$$

where the first part calculates the log loss of the ranking accuracy, the second part is the $l2$ norm of the parameters for regularization, and λ is the coefficient of the regularization term for trade-off.

We optimize the constrained loss function based on the Lagrange multipliers method [1, 4] in Equation 5. Let

$$\begin{aligned} \mathcal{L}(\mathbf{T},\mathbf{w},b,\mathbf{A}) = &\sum_{\forall(\mathbf{d}_i,\mathbf{d}_j),i\neq j} \log\left(1+e^{-y_{i,j}\left(\mathbf{w}^\top\mathbf{T}^\top(\mathbf{d}_i-\mathbf{d}_j)+b\right)}\right)+ \\ &\frac{\lambda}{2}||\mathbf{w}||^2+\sum_{i,j=1,\ldots,k\wedge i\neq j}\alpha_{i,j}\mathbf{t}_i^\top\mathbf{t}_j+\sum_{i=1}^{k}\alpha_{i,i}\left(1-\mathbf{t}_i^\top\mathbf{t}_i\right), \end{aligned} \tag{6}$$

where $\mathbf{A}$ is a matrix with k columns and k rows, and elements $\alpha_{i,j}$. Then, the optimum $\mathbf{T}$, $\mathbf{w}$ and b for minimizing $\mathcal{L}$ are the exact results of Equation 5.

In Phase I, we utilize gradient descent to generate the training dataset and the transformation matrix. Initially, we assign all 1s to the vector $\mathbf{w}_{k\times 1}$ so that all of the generated features in the ranking model have the same initial weight. We initialize the transformation matrix $\mathbf{T}$ in a random manner, following work on matrix generalization problems like matrix factorization-based collaborative filtering [25]. After initialization, the weight vector $\mathbf{w}$ and the factorized matrix can be updated iteratively with gradient descent until reaching convergence or the maximum number of iterations with the given learning rate. The gradients of the function $\mathcal{L}$ with respect to the variables are calculated as follows:

$$\begin{aligned} \nabla_{\mathbf{w}}\mathcal{L} &= \sum_{\forall(\mathbf{d}_i,\mathbf{d}_j),i\neq j}\frac{-y_{i,j}\mathbf{T}^\top(\mathbf{d}_i-\mathbf{d}_j)}{1+e^{y_{i,j}\left(\mathbf{w}^\top\mathbf{T}^\top(\mathbf{d}_i-\mathbf{d}_j)+b\right)}}+\lambda\mathbf{w} \\ \nabla_{\mathbf{t}_l}\mathcal{L} &= \sum_{\forall(\mathbf{d}_i,\mathbf{d}_j),i\neq j}\frac{-y_{i,j}w_l(\mathbf{d}_i-\mathbf{d}_j)}{1+e^{y_{i,j}\left(\mathbf{w}^\top\mathbf{T}^\top(\mathbf{d}_i-\mathbf{d}_j)+b\right)}}+ \\ &\quad \sum_{i\neq l}(\alpha_{l,i}+\alpha_{i,l})\mathbf{t}_i-2\alpha_{l,l}\mathbf{t}_l, \quad l=1,\ldots,k \\ \frac{\partial\mathcal{L}}{\partial b} &= \sum_{\forall(\mathbf{d}_i,\mathbf{d}_j),i\neq j}\frac{-y_{i,j}}{1+e^{y_{i,j}\left(\mathbf{w}^\top\mathbf{T}^\top(\mathbf{d}_i-\mathbf{d}_j)+b\right)}} \\ \frac{\partial\mathcal{L}}{\partial\alpha_{i,j}} &= \begin{cases}\mathbf{t}_i^\top\mathbf{t}_j, & i\neq j\\ 1-\mathbf{t}_i^\top\mathbf{t}_j, & i=j\end{cases} \quad \forall i,j=1,\ldots,k, \end{aligned} \tag{7}$$

where $\mathbf{t}_1, \mathbf{t}_2, \ldots \mathbf{t}_n$ are the column vectors of $\mathbf{T}$. Since gradient descent generally does not work with Lagrange multipliers, we use the basic differential multiplier method (BDMM) [41] for optimization, where the sign inversion for α in Equation 8 makes the optimization stable. Given a learning rate η, the update formulas of the gradient descent method are:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L} \\ \mathbf{t}_l &\leftarrow \mathbf{t}_l - \eta \nabla_{\mathbf{t}_l} \mathcal{L}, \text{ for } l = 1, \ldots, k \\ b &\leftarrow b - \eta \frac{\partial \mathcal{L}}{\partial b} \\ \alpha_{i,j} &\leftarrow \alpha_{i,j} + \eta \frac{\partial \mathcal{L}}{\partial \alpha_{i,j}}, \text{ for } i, j = 1, \ldots, k. \end{aligned} \tag{8}$$

4.2 Phase II: Generation of low-rank datasets

In LifeRank, Phase II generates all of the datasets for learning to rank, including the training, validation and test datasets. According to Definition 3, for each original matrix $\mathbf{X}$, the generated matrix $\mathbf{X}'$ can be obtained as a product of the original dataset $\mathbf{X}$ and the transformation matrix $\mathbf{T}$, formally $\mathbf{X}' = \mathbf{X}\mathbf{T}$. Then, the new low-dimensional dataset $\mathcal{X}'$ can be generated by integrating matrix $\mathbf{X}'$ with other information $\mathbf{I}_X$ that was filtered in the preprocessing step in Phase I.

4.3 Pseudocode

The pseudocode of LifeRank as a dimension reduction algorithm for ranking is summarized in Algorithm 1. Given the number of generated features k and a set of standard learning to rank datasets, including a training dataset $\mathcal{X}$, a validation dataset $\mathcal{V}$ and a test dataset $\mathcal{E}$, LifeRank tries to output new low-dimensional datasets $\mathcal{X}'$, $\mathcal{V}'$ and $\mathcal{E}'$ for training, validation and test, respectively, for the learning to rank procedure.

Algorithm 1 implements the two phases of LifeRank: (I) Lines 1–8 generate the transformation matrix $\mathbf{T}$ based on the original training dataset $\mathcal{X}$; (II) Using $\mathbf{T}$, lines 9–10 generate the low-dimensional matrices for training $\mathbf{X}'$, validation $\mathbf{V}'$ and test $\mathbf{E}'$. Then, line 11 constructs the low-dimensional training, validation and test datasets by directly integrating the low-rank matrices and their corresponding information filtered in the preprocessing steps in lines 1 and 9.

4.4 Discussion

In this section, we reveal a connection between the feature selection for ranking problem and the linear feature extraction for ranking problem. In particular, from the perspective of linear transformations of matrices, the feature selection for ranking problem can be defined as in Definition 4.

Algorithm 1: LifeRank: A Linear Feature Extraction Algorithm for Ranking

Input: A training dataset $\mathcal{X}$, a validation dataset $\mathcal{V}$, a test dataset $\mathcal{E}$, the learning rate η, and the number of features k in the set of generated document.
Output: A generated training dataset $\mathcal{X}'$, validation dataset $\mathcal{V}'$, and test dataset $\mathcal{E}'$, each with k features.

// Phase I
$(\mathbf{X}, \mathbf{I}_X) \leftarrow$ Preprocess$(\mathcal{X})$;
$\mathbf{T}, \mathbf{w}, \{\alpha_{i,j}\}_{i,j=1,\ldots,k} \leftarrow$ Initialize$(\mathbf{X}, k)$;
repeat
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}$;
  $\mathbf{t}_l \leftarrow \mathbf{t}_l - \eta \nabla_{\mathbf{t}_l} \mathcal{L}$, for $l = 1, \ldots, k$;
  $b \leftarrow b - \eta \frac{\partial \mathcal{L}}{\partial b}$;
  $\alpha_{i,j} \leftarrow \alpha_{i,j} + \eta \frac{\partial \mathcal{L}}{\partial \alpha_{i,j}}$, for $i, j = 1, \ldots, k$;
until *Reach convergence or the max iteration*;
// Phase II
$(\mathbf{V}, \mathbf{I}_V), (\mathbf{E}, \mathbf{I}_E) \leftarrow$ Preprocess$(\mathcal{V}, \mathcal{E})$;
$\mathbf{X}' \leftarrow \mathbf{X}\mathbf{T}, \quad \mathbf{V}' \leftarrow \mathbf{V}\mathbf{T}, \quad \mathbf{E}' \leftarrow \mathbf{V}\mathbf{T}$;
$\mathcal{X}', \mathcal{V}', \mathcal{E}' \leftarrow$ GenerateDatasets$(\mathbf{X}', \mathbf{V}', \mathbf{E}', \mathbf{I}_X, \mathbf{I}_V, \mathbf{I}_E)$;

Definition 4 (Feature selection for ranking) *Let $\mathbf{X}_{m\times n}$ be a document matrix, where the transpose of each row ${\mathbf{x}_i}^\top = \mathbf{d}_i$ is an n-dimensional vector, representing a document with n features. Feature selection for ranking aims to optimize a transformation matrix $\mathbf{T}_{n\times k}$ by solving the following optimization problem, so that a new document matrix $\mathbf{X}'_{m\times k} = \mathbf{X}_{m\times n}\mathbf{T}_{n\times k}$ can be generated, where each n-dimensional document vector $\mathbf{d}_i$ can be projected into a k-dimensional vector $\mathbf{d}'_i = \mathbf{T}^\top \mathbf{d}_i$:*

$$\arg\min_{\mathbf{T}} L(g(\mathbf{X}\mathbf{T})) \text{ such that } \begin{cases} \forall t_{i,j} \in \mathbf{T} : t_{i,j} = \{0, 1\} \\ \mathbf{T}^\top \mathbf{T} = \mathbf{I}. \end{cases} \tag{9}$$

Based on the optimized mapping function g, the new low-rank matrix can be generated as a product of the original matrix and the transformation matrix, i.e., $\mathbf{X}' = \mathbf{X}\mathbf{T}$.

The k columns of the transformation $\mathbf{T}$ mentioned in Defintion 4 present the k iterations of the feature selection processes. The constraints in Equation 9 guarantee that there is only one "1" in each column of the transformation matrix $\mathbf{T}$ and the others are all "0," indicating that each feature selection process only selects one feature. The second constraint $\mathbf{T}^\top \mathbf{T} = \mathbf{I}$ guarantees that the position of the unique "1" in each column is different from other columns, which is the index of the selected feature in that step.

Since the elements in the transformation matrix $\mathbf{T}$ can be any real numbers in Definition 3 while they are only either 0 or 1 in Definition 4, Definition 3 generalizes Definition 4, i.e., the problem of linear feature extraction for ranking generalizes the problem of feature selection for ranking. Because of this, linear feature extraction is expected to outperform or be at least as good as any feature selection technique. The linear feature extraction is expected to use more computational resources than feature selection, since former deals with the search space in real numbers and the

latter with binary case. However, this computational overhead is the tradeoff for the higher performance expected to be achieved by the extracted features, when utilized for learning to rank.

5 Experimental setup

5.1 Research questions

We list the research questions that guide the remainder of the paper.

RQ1 What is the performance of LifeRank in generating low-dimensional datasets? Does LifeRank outperform state-of-the-art feature selection algorithms? (See §6.1)

RQ2 Can the importance and redundancy of the features generated by LifeRank outperform those selected by feature selection algorithms? (See §6.2)

RQ3 What is the effect of the orthonormality constraints of the transformation matrix in Equation 4? Does it help enhance the performance of ranking predictions? (See §6.3.)

5.2 Datasets

In this study, we use the MQ2007 and MQ2008 datasets from LETOR 4.0 [42] and OHSUMED from LETOR 3.0 [43] to evaluate our algorithm. The LETOR[1] datasets are commonly used benchmarks in learning to rank. LETOR 4.0 is the latest version, which was released in July 2009. It uses the Gov2 web page collection (∼25M pages) and two query sets from the Million Query track of TREC 2007 and TREC 2008, which are referred to as MQ2007 and MQ2008. We use both MQ2007 and MQ2008 in our experiments. In MQ2007, there are about 1700 queries and about 70,000 query-document pairs, while MQ2008 has 800 queries and about 15,000 query-document pairs for training, validation and testing. In both datasets, each query-document pair has 46 features. We also use the OHSUMED dataset from LETOR 3.0, which was released in December 2008. OHSUMED is extracted from the online medical information database MEDLINE. It contains 106 queries and about 16,000 query-document pairs, where each query-document pair has 45 features.

In all the datasets that we use, relevance of documents with respect to queries is judged at three levels: 2 (definitely relevant), 1 (partially relevant), and 0 (not relevant). In our experiments, we use 5-fold cross validation. In each fold, 60% queries are used for training, 20% for validation and and the remaining 20% for testing. The performance numbers reported are averaged over the five folds.

5.3 Baselines

LifeRank aims to generate low-dimensional datasets for ranking. In this paper, we utilize three baselines to evaluate the datasets generated by our algorithm:

[1] http://research.microsoft.com/en-us/um/beijing/projects/letor/

Original datasets: We firstly use the original LETOR datasets as our first baseline, on which no selection or generation has been performed.

Datasets generated by GAS: GAS [16] incorporates importance and similarity information of the features into ranking. It greedily selects a subset of features by maximizing the total importance scores meanwhile minimizing the total similarity scores.

Datasets generated by FSMRank: FSMRank [26] trains a feature selection model with machine learning, which can select a subset of features meanwhile minimizing the ranking errors.

We then run Linear Regression [30]-based learning to rank and RankSVM[2] [21] to determine how well these datasets can address the ranking problem. The former makes pointwise predictions on the relevance of the documents by linear regression, which is implemented in the RankLib learning to rank toolkit.[3] The latter predicts pairwise ranking relation between each pair of documents directly by support vector machine (SVM). These are classical pointwise and pairwise learning to rank algorithms, respectively, with which we can clearly demonstrate the effects of dimension reduction.

Since LifeRank uses a linear approach for feature extraction, it is expected to show effectiveness mainly for linear learning-to-rank methods. This is the reason why we have chosen SVMRank and Linear Regression for experimentation.

5.4 Evaluation measures

5.4.1 Measures for ranking

We use two standard ranking accuracy metrics to evaluate the rankings generated by learning to rank algorithms: mean average precision (*MAP*) [3] and normalized discount cumulative gain (*NDCG@n*) [19].

Statistical significance of observed differences between the performance of two runs is tested using a two-tailed paired t-test and is denoted using $^{\blacktriangle}$ (or $^{\blacktriangledown}$) for strong significance for $\alpha = 0.01$; or $^{\triangle}$ (or $^{\triangledown}$) for weak significance for $\alpha = 0.05$.

5.4.2 Measures for features

We consider two metrics to evaluate the quality of the features: importance and redundancy.

The *importance* of each feature can be evaluated by the ranking performance when the feature is used as a ranking model to order the documents. In particular, we use *NDCG@5* for evaluation. Since for calculating these measures, for some features larger values correspond to higher ranks while for others smaller values lead to higher ranks, we utilize the strategy in GAS [16] for evaluation: We order the documents twice in ascending and descending manners respectively, and take the larger score as

[2] https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html
[3] https://sourceforge.net/p/lemur/wiki/RankLib/

the importance score of the features. Then we calculate the average importance of the features as the importance of the set of features $F = \{f_1, f_2, \ldots, f_k\}$:

$$Imp(F) = \frac{1}{k} \sum_{i=1}^{k} \max \left\{ eva(\mathcal{X}, f_i), eva(\mathcal{X}, -f_i) \right\},$$

where the function $eva(\mathcal{X}, f_i)$ returns the evaluation results of the ranking model f_i on the dataset $\mathcal{X}$.

The *redundancy* of features can be defined as the average similarity between each pair of features. In practice, we regard each feature as a ranking model to order the documents, and then calculate the similarity between each pair of features as the average similarity of their document rankings associated to different queries. Let Q be the set of queries in the given dataset, each associated with a set of documents for ranking. The redundancy of the features $F = \{f_1, f_2, \ldots, f_k\}$ is calculated as follows:

$$Rdd(F) = \frac{2}{k(k-1)} \sum_{f_i, f_j \in F, i>j} \frac{1}{|Q|} \sum_{q \in Q} sim \left(\sigma_i^{(q)}, \sigma_j^{(q)} \right),$$

where $\sigma_i^{(q)}$ is the ranking of the document associated to the query q when the feature f_i is used as the ranking model to order the documents. In this paper, we take the absolute value of Kendall's τ correlation coefficient [24] as the similarity metric for rankings:

$$\tau \left(\sigma_i, \sigma_j \right) = \frac{N_c - N_d}{N_c + N_d}, \tag{10}$$

where N_c and N_d are the numbers of the concordant pairs and discordant pairs respectively between rankings σ_i and σ_j.

The range of $\tau \left(\sigma_i, \sigma_j \right)$ is $[-1, 1]$, where the sign indicates that the correlation between σ_i and σ_j is either positive or negative, and the absolute value indicates the strength of the correlation. Since positive and negative values should not neutralize and we only consider the strength of the correlations, we take the absolute value of Kendall's τ as the similarity metric in the definition of redundancy.

6 Experimental results

6.1 Performance on generated datasets

Tables 1, 2 and 3 list the results obtained in our experiments on the MQ2007, MQ2008 and OHSUMED datasets, respectively. They show the NDCG@1–10 and MAP scores for the RankSVM and Linear Regression learning to rank algorithms on 4 categories of datasets: the original datasets and 3 datasets generated by dimension reduction algorithms including GAS, FSMRank and our LifeRank. For each dimension reduction algorithm, we consider k generated features, with $k = 5$, 10, 15, 20. The results for the original dataset in the tables are independent of the value of k, but are repeated nevertheless for ease of comparison. The values in bold represent the best performance among GAS, FSMSVM and LifeRank.

Table 1: Ranking performance on MQ2007 and selected/generated datasets.
Statistical significance shown for LifeRank against Original, GAS and FSMSVM
Performance for Original is independent of value of k (corresponds to original dataset)

Performance for RankSVM

NDCG	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10	MAP
k = 5											
Original	0.4079	0.4007	0.4009	0.4030	0.4077	0.4129	0.4201	0.4275	0.4336$^{\triangledown}$	0.4391$^{\triangledown}$	0.4615$^{\blacktriangledown}$
GAS	0.3751	0.3807	0.3869$^{\triangle}$	0.3946	0.3980	0.4068	0.4113	0.4177	0.4242	0.4290	**0.4563**
FSMSVM	0.3598$^{\blacktriangle}$	0.3703$^{\blacktriangle}$	0.3773$^{\blacktriangle}$	0.3811$^{\blacktriangle}$	0.3871$^{\blacktriangle}$	0.3941$^{\blacktriangle}$	0.3997$^{\blacktriangle}$	0.4061$^{\blacktriangle}$	0.4129$^{\blacktriangle}$	0.4193$^{\blacktriangle}$	0.4512
LifeRank	**0.3925**	**0.3925**	**0.3975**	**0.4009**	**0.4062**	**0.4118**	**0.4162**	**0.4209**	**0.4256**	**0.4312**	0.4539
k = 10											
Original	0.4079	0.4007	0.4009	0.4030$^{\triangle}$	0.4077$^{\triangle}$	0.4129$^{\blacktriangle}$	0.4201$^{\triangle}$	0.4275	0.4336	0.4391	0.4615
GAS	0.3897	0.3893$^{\blacktriangle}$	0.3914$^{\blacktriangle}$	0.3965$^{\blacktriangle}$	0.4029$^{\blacktriangle}$	0.4098$^{\blacktriangle}$	0.4153$^{\blacktriangle}$	0.4209$^{\blacktriangle}$	0.4272$^{\blacktriangle}$	0.4343$^{\blacktriangle}$	0.4558$^{\triangle}$
FSMSVM	0.3919	0.3917	0.3982$^{\triangle}$	0.4027$^{\triangle}$	0.4079$^{\triangle}$	0.4134$^{\blacktriangle}$	0.4187$^{\blacktriangle}$	0.4239$^{\blacktriangle}$	0.4290$^{\blacktriangle}$	0.4349$^{\blacktriangle}$	0.4593
LifeRank	**0.4037**	**0.4023**	**0.4089**	**0.4117**	**0.4161**	**0.4215**	**0.4264**	**0.4312**	**0.4370**	**0.4423**	**0.4634**
k = 15											
Original	0.4079	0.4007	0.4009	0.4030	0.4077	0.4129	0.4201	0.4275	0.4336	0.4391	0.4615
GAS	0.3942	0.3954	0.3998	0.4023	0.4063$^{\blacktriangle}$	0.4126$^{\triangle}$	0.4195	0.4256$^{\triangle}$	0.4316$^{\blacktriangle}$	0.4372$^{\blacktriangle}$	0.4593$^{\blacktriangle}$
FSMSVM	0.3905	0.3937$^{\triangle}$	0.4005	0.4060	0.4106	0.4144	0.4201	0.4250$^{\triangle}$	0.4309$^{\blacktriangle}$	0.4365$^{\blacktriangle}$	0.4589$^{\triangle}$
LifeRank	**0.3972**	**0.4032**	**0.4061**	**0.4082**	**0.4141**	**0.4194**	**0.4242**	**0.4308**	**0.4376**	**0.4436**	**0.4635**
k = 20											
Original	0.4079	0.4007	0.4009	0.4030$^{\triangle}$	0.4077$^{\triangle}$	0.4129$^{\blacktriangle}$	0.4201$^{\triangle}$	0.4275	0.4336	0.4391	0.4615
GAS	0.4014	0.3967$^{\blacktriangle}$	0.4007	0.4027$^{\blacktriangle}$	0.4088$^{\blacktriangle}$	0.4137$^{\blacktriangle}$	0.4189$^{\blacktriangle}$	0.4257$^{\blacktriangle}$	0.4326$^{\triangle}$	0.4394$^{\triangle}$	0.4601$^{\blacktriangle}$
FSMSVM	0.3882$^{\blacktriangle}$	0.3929$^{\blacktriangle}$	0.4004	0.4060	0.4096	0.4138$^{\triangle}$	0.4203$^{\triangle}$	0.4259	0.4310$^{\triangle}$	0.4368$^{\blacktriangle}$	0.4595$^{\triangle}$
LifeRank	**0.4097**	**0.4077**	**0.4058**	**0.4106**	**0.4153**	**0.4213**	**0.4265**	**0.4311**	**0.4374**	**0.4438**	**0.4640**

Performance for Linear Regression

NDCG	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10	MAP
k = 5											
Original	0.3750	0.3854	0.3882	0.3926	0.3979	0.4034	0.4088	0.4154	0.4208	0.4277	0.4497
GAS	0.3712	0.3751	0.3797$^{\triangle}$	0.3851$^{\triangle}$	0.3894	0.3952	0.4011$^{\triangle}$	0.4071$^{\triangle}$	0.4136$^{\triangle}$	0.4196$^{\triangle}$	0.4462
FSMSVM	0.3554$^{\blacktriangle}$	0.3634$^{\blacktriangle}$	0.3673$^{\blacktriangle}$	0.3741$^{\blacktriangle}$	0.3780$^{\blacktriangle}$	0.3872$^{\blacktriangle}$	0.3929$^{\blacktriangle}$	0.4002$^{\blacktriangle}$	0.4076$^{\blacktriangle}$	0.4145$^{\blacktriangle}$	0.4489
LifeRank	**0.3852**	**0.3874**	**0.3908**	**0.3955**	**0.3962**	**0.4026**	**0.4089**	**0.4149**	**0.4213**	**0.4279**	**0.4507**
k = 10											
Original	0.3750	0.3854	0.3882	0.3926	0.3979	0.4034	0.4088	0.4154	0.4208	0.4277	0.4497
GAS	0.3886	0.3879	0.3881	0.3929	0.3980	0.4031	0.4090	0.4146	0.4198	0.4261	0.4491
FSMSVM	**0.3903**	**0.3951**	**0.3936**	0.3950	0.3991	0.4049	0.4111	0.4166	0.4223	0.4285	0.4494
LifeRank	0.3852	0.3920	0.3926	**0.3976**	**0.4026**	**0.4073**	**0.4146**	**0.4206**	**0.4270**	**0.4312**	**0.4507**
k = 15											
Original	0.3750	0.3854	0.3882	0.3926	0.3979	0.4034	0.4088$^{\triangle}$	0.4154	0.4208$^{\triangle}$	0.4277$^{\triangle}$	0.4497
GAS	0.3767	0.3849	0.3913	0.3954	0.4000	0.4070	0.4133	0.4191	0.4250	0.4315	**0.4528**
FSMSVM	0.3800	0.3825	0.3855	0.3882$^{\blacktriangle}$	0.3922$^{\blacktriangle}$	0.3976$^{\blacktriangle}$	0.4035$^{\blacktriangle}$	0.4091$^{\blacktriangle}$	0.4147$^{\blacktriangle}$	0.4215$^{\blacktriangle}$	0.4441$^{\blacktriangle}$
LifeRank	**0.3842**	**0.3888**	**0.3943**	**0.3984**	**0.4019**	**0.4091**	**0.4161**	**0.4207**	**0.4274**	**0.4336**	0.4513
k = 20											
Original	0.3750	0.3854	0.3882	0.3926	0.3979	0.4034	0.4088	0.4154	0.4208	0.4277	0.4497
GAS	0.3783	**0.3897**	**0.3956**	**0.3990**	**0.4021**	**0.4079**	0.4120	0.4178	**0.4255**	**0.4313**	**0.4521**
FSMSVM	0.3742	0.3867	0.3916	0.3937	0.3970	0.4011	0.4062$^{\triangle}$	0.4132	0.4184	0.4244$^{\triangle}$	0.4483
LifeRank	**0.3828**	0.3894	0.3912	0.3948	0.4014	0.4068	**0.4121**	**0.4184**	0.4243	0.4306	0.4514

Table 2: Ranking performance on MQ2008 and selected/generated datasets.
Statistical significance shown for LifeRank against Original, GAS and FSMSVM
Performance for Original is independent of value of k (corresponds to original dataset)

Performance for RankSVM											
NDCG	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10	MAP
k = 5											
Original	0.3712	0.3933$^{\blacktriangle}$	0.4238$^{\triangle}$	0.4485$^{\triangle}$	0.4672$^{\blacktriangle}$	0.4814$^{\triangle}$	0.4875$^{\triangle}$	0.4531$^{\triangle}$	0.2234	0.2284	0.4714
GAS	0.3678	0.3983$^{\triangle}$	0.4213$^{\blacktriangle}$	0.4492$^{\triangle}$	0.4665$^{\blacktriangle}$	0.4800$^{\blacktriangle}$	0.4862$^{\blacktriangle}$	0.4522$^{\blacktriangle}$	0.2207$^{\blacktriangle}$	0.2246$^{\blacktriangle}$	0.4714
FSMSVM	**0.3780**	0.4126	0.4384	0.4599	0.4761	0.4909	0.4968	0.4616	**0.2284**	0.2326	0.4776
LifeRank	0.3767	**0.4168**	**0.4389**	**0.4606**	**0.4806**	**0.4921**	**0.4976**	**0.4627**	0.2280	**0.2329**	**0.4788**
k = 10											
Original	0.3712	0.3933$^{\blacktriangle}$	0.4238$^{\triangle}$	0.4485$^{\triangle}$	0.4672$^{\blacktriangle}$	0.4814	0.4875$^{\triangle}$	0.4531$^{\blacktriangle}$	0.2234	0.2284$^{\triangle}$	0.4714
GAS	0.3698	0.4015$^{\triangle}$	0.4292	0.4565	0.4732	0.4862	0.4929	0.4551$^{\triangle}$	0.2217$^{\triangle}$	0.2266$^{\blacktriangle}$	0.4776
FSMSVM	0.3759	0.4157	0.4371	0.4589	0.4781	**0.4925**	0.4962	0.4617	0.2276	0.2322	**0.4793**
LifeRank	**0.3763**	**0.4181**	**0.4384**	**0.4612**	**0.4796**	0.4900	**0.4972**	**0.4625**	**0.2281**	**0.2345**	0.4792
k = 15											
Original	0.3712	0.3933$^{\blacktriangle}$	0.4238$^{\blacktriangle}$	0.4485$^{\blacktriangle}$	0.4672$^{\blacktriangle}$	0.4814$^{\blacktriangle}$	0.4875$^{\blacktriangle}$	0.4531$^{\blacktriangle}$	0.2234$^{\blacktriangle}$	0.2284$^{\blacktriangle}$	0.4714$^{\triangle}$
GAS	0.3720	0.3984$^{\triangle}$	0.4320	0.4533$^{\triangle}$	0.4711$^{\triangle}$	0.4851	0.4902$^{\triangle}$	0.4543$^{\blacktriangle}$	0.2223$^{\blacktriangle}$	0.2279$^{\blacktriangle}$	0.4742
FSMSVM	**0.3788**	**0.4165**	0.4351	0.4557	0.4761	0.4905	0.4967	0.4613	0.2265	0.2311	0.4788
LifeRank	0.3771	0.4140	**0.4379**	**0.4622**	**0.4804**	**0.4920**	**0.4972**	**0.4633**	**0.2310**	**0.2349**	**0.4792**
k = 20											
Original	0.3712	0.3933	0.4238$^{\triangle}$	0.4485$^{\blacktriangle}$	0.4672$^{\triangle}$	0.4814$^{\triangle}$	0.4875$^{\blacktriangle}$	0.4531$^{\blacktriangle}$	0.2234$^{\triangle}$	0.2284$^{\triangle}$	0.4714
GAS	0.3656	0.4027	0.4313	0.4527$^{\triangle}$	0.4720	0.4850	0.4921	0.4566	0.2254	0.2298	0.4719
FSMSVM	**0.3737**	**0.4144**	0.4343	0.4588	0.4757	0.4902	0.4946	0.4590	0.2249	0.2302	**0.4754**
LifeRank	0.3712	0.4045	**0.4363**	**0.4619**	**0.4763**	**0.4903**	**0.4971**	**0.4617**	**0.2293**	**0.2338**	0.4751
Performance for Linear Regression											
NDCG	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10	MAP
k = 5											
Original	0.3465	0.3617$^{\blacktriangle}$	0.3961	0.4225	0.4407	0.4558$^{\triangle}$	0.4684$^{\triangle}$	0.4746$^{\blacktriangle}$	0.4806$^{\blacktriangle}$	0.4871$^{\blacktriangle}$	0.4550$^{\blacktriangle}$
GAS	0.3537	0.3691$^{\triangle}$	0.3947	0.4184	0.4390	0.4584	0.4692	0.4776	0.4838	0.4892	0.4630
FSMSVM	0.3541	0.3755$^{\triangle}$	0.4002	**0.4191**	0.4396$^{\triangle}$	0.4563$^{\blacktriangle}$	0.4681$^{\triangle}$	0.4791$^{\triangle}$	0.4842$^{\triangle}$	0.4903$^{\triangle}$	0.4593$^{\triangle}$
LifeRank	**0.3691**	**0.3907**	**0.4089**	0.3955	**0.4491**	**0.4664**	**0.4772**	**0.4864**	**0.4924**	**0.4979**	**0.4685**
k = 10											
Original	0.3465$^{\triangle}$	0.3617$^{\blacktriangle}$	0.3961$^{\triangle}$	0.4225	0.4407$^{\triangle}$	0.4558$^{\triangle}$	0.4684$^{\blacktriangle}$	0.4746$^{\blacktriangle}$	0.4806$^{\blacktriangle}$	0.4871$^{\blacktriangle}$	0.4550$^{\blacktriangle}$
GAS	0.3605	0.3804	0.3990	**0.4282**	0.4490	0.4628	0.4742	0.4840	0.4895	0.4942	0.4669
FSMSVM	0.3512	0.3779	0.4017	0.4213	0.4442	0.4601	0.4702$^{\triangle}$	0.4789$^{\triangle}$	0.4847	0.4910	0.4622
LifeRank	**0.3698**	**0.3857**	**0.4085**	0.3976	**0.4501**	**0.4659**	**0.4791**	**0.4859**	**0.4908**	**0.4970**	**0.4686**
k = 15											
Original	0.3465	0.3617$^{\blacktriangle}$	0.3961	0.4225	0.4407	0.4558$^{\triangle}$	0.4684	0.4746$^{\blacktriangle}$	0.4806$^{\triangle}$	0.4871$^{\triangle}$	0.4550$^{\triangle}$
GAS	**0.3652**	0.3795	**0.4114**	**0.4302**	0.4497	0.4641	0.4757	0.4845	**0.4914**	**0.4964**	**0.4686**
FSMSVM	0.3631	0.3822	0.4053	0.4285	**0.4527**	**0.4669**	**0.4774**	0.4850	0.4901	0.4956	0.4647
LifeRank	0.3618	**0.3842**	0.4032	0.4296	0.4482	0.4657	0.4766	**0.4852**	0.4904	0.4955	0.4662
k = 20											
Original	0.3465$^{\blacktriangle}$	0.3617$^{\blacktriangle}$	0.3961	0.4225	0.4407$^{\triangle}$	0.4558	0.4684	0.4746$^{\triangle}$	0.4806$^{\blacktriangle}$	0.4871$^{\blacktriangle}$	0.4550$^{\blacktriangle}$
GAS	0.3588	0.3801	**0.4100**	**0.4300**	0.4486	0.4650	0.4744	**0.4843**	**0.4906**	**0.4961**	0.4660
FSMSVM	0.3601$^{\triangle}$	0.3803	0.4075	0.4266	0.4525	**0.4668**	**0.4772**	0.4833	0.4889	0.4945	0.4661
LifeRank	**0.3712**	**0.3820**	0.4018	0.3948	**0.4495**	0.4628	0.4748	0.4833	0.4897	0.4956	**0.4662**

Table 3: Ranking performance on OHSUMED and selected/generated datasets.
Statistical significance shown for LifeRank against Original, GAS and FSMSVM
Performance for Original is independent of value of k (corresponds to original dataset)

Performance for RankSVM

NDCG	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10	MAP
					k = 5						
Original	0.5416	0.5076	0.4775	0.4565$^{\triangle}$	0.4439$^{\blacktriangle}$	0.4452	0.4433	0.4405	0.4338$^{\blacktriangle}$	0.4300$^{\blacktriangle}$	0.4345$^{\blacktriangle}$
GAS	0.5332	0.4901	0.4739	0.4630$^{\triangle}$	0.4578$^{\blacktriangle}$	0.4503$^{\triangle}$	0.4432$^{\blacktriangle}$	0.4398$^{\triangle}$	0.4374$^{\blacktriangle}$	0.4340$^{\blacktriangle}$	0.4642$^{\blacktriangledown}$
FSMSVM	**0.5771**$^{\triangledown}$	0.4889	0.4772	0.4749	0.4694$^{\triangle}$	0.4609	0.4622	0.4559	0.4529	0.4518	**0.4728**
LifeRank	0.5170	**0.5000**	**0.5015**	**0.4950**	**0.4905**	**0.4749**	**0.4701**	**0.4628**	**0.4684**	**0.4668**	0.4523
					k = 10						
Original	0.5416	0.5076	0.4775	0.4565$^{\blacktriangle}$	0.4439$^{\blacktriangle}$	0.4452$^{\blacktriangle}$	0.4433$^{\blacktriangle}$	0.4405$^{\blacktriangle}$	0.4338$^{\blacktriangle}$	0.4300$^{\blacktriangle}$	0.4345$^{\blacktriangle}$
GAS	**0.5677**	**0.5390**	0.5088	0.4944	0.4873	0.4673	0.4652	0.4605	0.4549	0.4507$^{\triangle}$	0.4466
FSMSVM	0.5296	0.4866$^{\blacktriangle}$	0.4794$^{\blacktriangle}$	0.4745$^{\blacktriangle}$	0.4636$^{\blacktriangle}$	0.4602$^{\triangle}$	0.4558$^{\blacktriangle}$	0.4531$^{\triangle}$	0.4484$^{\blacktriangle}$	0.4463$^{\blacktriangle}$	0.4459
LifeRank	0.5518	0.5373	**0.5185**	**0.5053**	**0.4910**	**0.4811**	**0.4774**	**0.4699**	**0.4692**	**0.4663**	**0.4505**
					k = 15						
Original	0.5416	0.5076	0.4775	0.4565$^{\triangle}$	0.4439$^{\blacktriangle}$	0.4452$^{\triangle}$	0.4433	0.4405$^{\triangle}$	0.4338$^{\blacktriangle}$	0.4300$^{\blacktriangle}$	0.4345$^{\blacktriangle}$
GAS	0.5771	0.5068	0.4850	0.4713	0.4656	0.4552	0.4524	0.4494	0.4439$^{\triangle}$	0.4419$^{\blacktriangle}$	0.4402$^{\blacktriangle}$
FSMSVM	**0.5834**	0.5317	0.5021	0.4856	0.4723	0.4660	0.4606	0.4557	0.4525	0.4500$^{\triangle}$	0.4452$^{\blacktriangle}$
LifeRank	0.5769	**0.5344**	**0.5065**	**0.4942**	**0.4835**	**0.4713**	**0.4672**	**0.4630**	**0.4632**	**0.4628**	**0.4536**
					k = 20						
Original	0.5416	0.5076	0.4775$^{\triangle}$	0.4565$^{\blacktriangle}$	0.4439$^{\blacktriangle}$	0.4452$^{\blacktriangle}$	0.4433$^{\blacktriangle}$	0.4405$^{\blacktriangle}$	0.4338$^{\blacktriangle}$	0.4300$^{\blacktriangle}$	0.4345$^{\blacktriangle}$
GAS	0.5519	0.5051	0.4838$^{\triangle}$	0.4761$^{\triangle}$	0.4710$^{\blacktriangle}$	0.4520$^{\blacktriangle}$	0.4473$^{\blacktriangle}$	0.4463$^{\blacktriangle}$	0.4401$^{\blacktriangle}$	0.4405$^{\blacktriangle}$	0.4387$^{\blacktriangle}$
FSMSVM	0.5173$^{\triangle}$	0.4848$^{\triangle}$	0.4816$^{\triangle}$	0.4766	0.4642$^{\blacktriangle}$	0.4522$^{\blacktriangle}$	0.4452$^{\blacktriangle}$	0.4411$^{\blacktriangle}$	0.4388$^{\blacktriangle}$	0.4387$^{\blacktriangle}$	0.4441$^{\blacktriangle}$
LifeRank	**0.5805**	**0.5416**	**0.5204**	**0.5029**	**0.4976**	**0.4834**	**0.4765**	**0.4725**	**0.4669**	**0.4656**	**0.4519**

Performance for Linear Regression

NDCG	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10	MAP
					k = 5						
Original	0.4830	0.4800	0.4749	0.4548	0.4483	0.4368	0.4389	0.4343	0.4311	0.4302	0.4333
GAS	0.4762	0.4491	0.4489	0.4466	0.4378	0.4286	0.4275	0.4209	0.4201	0.4202	0.4549
FSMSVM	**0.5271**	**0.4852**	**0.4686**	**0.4589**	**0.4513**	0.4403	0.4342	0.4335	0.4306	0.4292	**0.4655**
LifeRank	0.4941	0.4736	0.4554	0.4586	0.4485	**0.4459**	**0.4425**	**0.4402**	**0.4347**	**0.4341**	0.4599
					k = 10						
Original	0.4830	0.4800	0.4749	0.4548	0.4483	0.4368	0.4389	0.4343	0.4311	0.4302	0.4333
GAS	0.5202	**0.4883**	**0.4680**	0.4597	0.4543	0.4460	0.4415	0.4354	0.4312	0.4275	0.4339
FSMSVM	0.5082	0.4556	0.4502	0.4395	0.4333	0.4245	0.4259	0.4223	0.4216	0.4180	0.4344
LifeRank	**0.5330**	0.4866	**0.4680**	**0.4681**	**0.4601**	**0.4494**	**0.4436**	**0.4423**	**0.4370**	**0.4366**	**0.4358**
					k = 15						
Original	0.4830	0.4800	0.4749	0.4548	0.4483	0.4368	0.4389	0.4343	0.4311	0.4302	0.4333
GAS	0.5105	**0.4941**	0.4791	0.4654	0.4510	0.4412	0.4351	0.4331	0.4303	0.4301	0.4298$^{\triangle}$
FSMSVM	0.4984	0.4611	0.4639	0.4567	0.4485	0.4400	0.4325	0.4321	0.4302	0.4282	**0.4405**
LifeRank	**0.5342**	0.4839	**0.5001**	**0.4781**	**0.4678**	**0.4529**	**0.4437**	**0.4412**	**0.4370**	**0.4353**	0.4399
					k = 20						
Original	0.4830	0.4800	0.4749	0.4548	0.4483	0.4368	0.4389	0.4343	0.4311	0.4302	0.4333
GAS	0.5050	0.4943$^{\triangle}$	0.4813$^{\blacktriangle}$	**0.4706**$^{\blacktriangledown}$	0.4546$^{\blacktriangle}$	**0.4497**$^{\triangledown}$	0.4389$^{\blacktriangle}$	0.4367$^{\blacktriangle}$	0.4324$^{\blacktriangle}$	0.4299$^{\blacktriangle}$	0.4308$^{\blacktriangle}$
FSMSVM	0.4984	0.4706	0.4574	0.4510	0.4489	0.4416	0.4370	0.4365	0.4341	0.4315	**0.4369**
LifeRank	**0.5264**	**0.5021**	**0.4882**	0.4686	**0.4571**	0.4467	**0.4439**	**0.4406**	**0.4359**	**0.434**1	**0.4369**

Overall, from the tables we can see that: (1) The performance of ranking algorithms can be maintained or slightly improved on the datasets generated by dimension reduction techniques. (2) The performance of the ranking algorithms on the datasets generated by LifeRank is higher than those generated by GAS and FSMRank in most cases. Let us now take a closer look.

6.1.1 Performance of RankSVM

For RankSVM, we can see that LifeRank clearly shows improvements over the original datasets for all the three benchmarks (MQ2007, MQ2008 and OHSUMED) in terms of NDCG@1–10 as well as MAP. The only exception is MQ2007 for $k = 5$, where the performance of LifeRank as well as the other generated datasets does not beat the original dataset. We can also see from the tables that LifeRank clearly outperforms other generated datasets (GAS and FSMSVM) on NDCG@1–10 for all the benchmarks and all values of k.

In terms of MAP, LifeRank outperforms the other generated datasets in most cases. The few exceptions include the case for MQ2007, when GAS has a higher MAP for $k = 5$. For MQ2008, FSMSVM attains slightly higher MAP score than LifeRank for $k = 10$ and $k = 20$, but these differences are not significant. Also, for OHSUMED when $k = 5$, the MAP score attained by LifeRank is lower than FSMSVM and GAS, but it is still an improvement over the original dataset.

6.1.2 Performance of Linear Regression

Also in the case of Linear Regression, for all three benchmarks (MQ2007, MQ2008 and OHSUMED) LifeRank clearly shows improvements over the original datasets in terms of NDCG@1–10 as well as MAP. The only exception is MQ2007 for $k = 5$, where the original dataset performs better than LifeRank as well as the other generated datasets.

On NDCG@1–10, for MQ2007 LifeRank gives the best performance for all values of k, except for $k = 20$, where GAS gives the best performance. For MQ2008, LifeRank gives the best performances for $k = 5$ and $k = 10$ on NDCG@1–10. However, for $k = 15$ and $k = 20$, there is mixed performance where all GAS, FSMSVM and LifeRank give best performances in certain cases. For, OHSUMED, LifeRank gives the best performance on NDCG@1–10 in most cases.

In terms of MAP, LifeRank gives the best performance for MQ2007 for $k = 5$ and $k = 10$, whereas for $k = 15$ and $k = 20$ the best performance is given by GAS. For MQ2008, LifeRank outperforms others for all values of k, except for $k = 15$ where the best performance is given by GAS. Moreover, for OHSUMED, FSMSVM outperforms the others for $k = 5$ and $k = 15$, while LifeRank gives the best performance for $k = 10$. In case of $k = 20$, there is a tie between LifeRank and FSMSVM.

6.1.3 Statistical Significance Overview

In Tables 1, 2 and 3, markups are provided to denote the statistical significance between LifeRank and the following baselines: original dataset, GAS and FSMSVM. It should be noted that the original dataset is independent of the values of k, but is repeated in the table to indicate statistical significance between it and datasets generated by LifeRank for different values of k.

It can be observed from Table 1 that for MQ2007 in the case of RankSVM, there is strong to weak significance between LifeRank and the baselines in most cases across the metrics, while there is no significance shown against original for $k = 15$. Moreover, for $k = 5$, significance is shown against original and GAS in few cases. For Linear Regression, there is strong significance shown against FSMSVM for $k = 5$ and $k = 15$, though there is not much significance shown for $k = 10$ and $k = 20$. Also, weak significance is shown against GAS in few cases for $k = 5$ and against original for $k = 15$.

Table 2 for MQ2008 shows no statistically significant differences between LifeRank and FSMSVM for RankSVM. There is weak to strong statistical significance for LifeRank against original dataset for most cases and against GAS mainly for $k = 5, 10$ and 15. For Linear Regression, LifeRank shows weak to strong statistical significance against original in most cases, GAS in no cases and FSMSVM in few cases. Moreover, Table 3 for OHSUMED shows statistical significance for RankSVM in many cases against the baselines, whereas there is statistical significance observed for Liner Regression for few cases. The comparative lack of statistical significance seen for MQ2008 and OHSUMED can most probably be attributed to the relatively small size of these datasets.

6.2 Quality of the generated features

Table 4 lists the quality scores of the features from four datasets: the original datasets and the three datasets generated by GAS, FSMRank and LifeRank, respectively, for $k = 10$.

Table 4: Performance of the generated features.

Datasets	MQ2007		MQ2008		OHSUMED	
	Imp	*Rdd*	*Imp*	*Rdd*	*Imp*	*Rdd*
Original	0.2671	0.4833	0.3297	0.5318	0.3763	0.5592
GAS	0.2643	**0.3242**	0.3235	**0.3308**	0.3603	**0.3904**
FSMRank	0.3005	0.4706	0.3723	0.5276	0.4170	0.5412
LifeRank	**0.3214**	0.4606	**0.4095**	0.5758	**0.4422**	0.8881

From the table we see that: (1) GAS can significantly reduce the redundancy of the features. The redundancy of the features selected by GAS is the lowest among the four datasets. However, the importance of the features selected by GAS is also lowest and even slightly lower than that of the original datasets. (2) FSMRank can improve

the importance of the features while reducing their redundancy, but the differences in terms redundancy are subtle. (3) LifeRank can sharply improve the importance of the features. The importance of the features generated by LifeRank is highest among the four datasets. Besides, the redundancy of the features can also be slightly reduced by LifeRank for MQ2007 but deteriorated for MQ2008 and OHSUMED. Worse redundancy for LifeRank in comparison with GAS and FSMSVM could be because of the reason that, while these baselines are feature selection methods, for LifeRank each extracted feature is a linear combination of the original features. Moreover, it can be observed that for the larger dataset MQ2007, redundancy for LifeRank is comparable to the baselines, and even better than FSMSVM. However, for smaller dataset MQ2008, the redundancy is worse than the baselines. For the smallest dataset OHSUMED, it is worse than the baselines by a greater difference.

6.3 Effect of the Orthonormality Constraints

To confirm that the orthonormality constraints used in LifeRank do indeed contribute to performance gains, we re-generated the datasets for the benchmarks MQ2007, MQ2008 and OHSUMED using LifeRank for $k = 10$, but this time without the incorporation of the constraints in its algorithm in Phase I (see Algorithm 1, line 1–8). Table 5 shows the comparison of performances of ranking algorithms, for datasets generated by LifeRank and LifeRank without orthonormality constraints (represented by LifeRank^{NO}). Moreover, markups are presented in the table to denote to the statistical significance between LifeRank and LifeRank^{NO}.

From the results in Table 5 we see that the datasets generated by LifeRank show significant improvements in performance over the datasets generated by LifeRank^{NO} for both learning to rank algorithms, RankSVM and Linear Regression. Performance gains can be observed on all three benchmarks and across all performance measures (NDCG@1–10 and MAP). Hence, these results show that the usage of orthonormality constraints is beneficial in the LifeRank algorithm. Also, strong statistical significance between LifeRank and LifeRank^{NO} can be observed for all three benchmarks for RankSVM as well as Linear Regression, across all performance measures, except for a small number of cases where weak or no statistical significance is seen.

7 Conclusion

In this paper, we have addressed the feature extraction problem for learning to rank, and have proposed LifeRank, a linear feature extraction algorithm for ranking. LifeRank regards each dataset for ranking as a matrix, referred to as the *original matrix*. We then optimize a *transformation matrix* by minimizing a classic pairwise learning to rank loss function, so that we can discover the optimal one that matches the ranking task. Then a new matrix (dataset) can be generated by the product of original matrix and transformation matrix. Extensive experiments on benchmark datasets show the performance gains of LifeRank in comparison with the state-of-the-art algorithms.

The performance of LifeRank has been evaluated for RankSVM and Linear Regression. In future work, its benefits for other learning to rank algorithms could be

Table 5: Effect of orthonormality constraints on datasets for $k = 10$.
Statistical significance shown for LifeRank against LifeRank^{NO}

Performance for RankSVM											
NDCG	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10	MAP
MQ2007											
LifeRank	**0.4037**$^{\blacktriangle}$	**0.4023**$^{\blacktriangle}$	**0.4089**$^{\blacktriangle}$	**0.4117**$^{\blacktriangle}$	**0.4161**$^{\blacktriangle}$	**0.4215**$^{\blacktriangle}$	**0.4264**$^{\blacktriangle}$	**0.4312**$^{\blacktriangle}$	**0.4370**$^{\blacktriangle}$	**0.4423**$^{\blacktriangle}$	**0.4634**$^{\blacktriangle}$
LifeRank^{NO}	0.3808	0.3881	0.3923	0.3939	0.3996	0.4044	0.4081	0.4148	0.4198	0.4269	0.4528
MQ2008											
LifeRank	**0.3763**	**0.4181**$^{\blacktriangle}$	**0.4384**$^{\triangle}$	**0.4612**$^{\blacktriangle}$	**0.4796**$^{\blacktriangle}$	**0.4900**$^{\triangle}$	**0.4972**$^{\triangle}$	**0.4625**$^{\triangle}$	**0.2281**$^{\blacktriangle}$	**0.2345**$^{\blacktriangle}$	**0.4792**$^{\triangle}$
LifeRank^{NO}	0.3618	0.3987	0.4264	0.4429	0.4657	0.4807	0.4887	0.4552	0.2199	0.2236	0.4704
OHSUMED											
LifeRank	0.5518	**0.5373**$^{\triangle}$	**0.5185**$^{\blacktriangle}$	**0.5053**$^{\blacktriangle}$	**0.4910**$^{\blacktriangle}$	**0.4811**$^{\triangle}$	**0.4774**$^{\blacktriangle}$	**0.4699**$^{\blacktriangle}$	**0.4692**$^{\blacktriangle}$	**0.4663**$^{\blacktriangle}$	**0.4505**$^{\blacktriangle}$
LifeRank^{NO}	**0.5703**	0.4900	0.4707	0.4673	0.4595	0.4522	0.4450	0.4399	0.4360	0.4312	0.4404
Performance for Linear Regression											
NDCG	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10	MAP
MQ2007											
LifeRank	**0.3852**$^{\blacktriangle}$	**0.3920**$^{\blacktriangle}$	**0.3926**$^{\blacktriangle}$	**0.3976**$^{\blacktriangle}$	**0.4026**$^{\blacktriangle}$	**0.4073**$^{\blacktriangle}$	**0.4146**$^{\blacktriangle}$	**0.4206**$^{\blacktriangle}$	**0.4270**$^{\blacktriangle}$	**0.4312**$^{\blacktriangle}$	**0.4507**$^{\blacktriangle}$
LifeRank^{NO}	0.3584	0.3691	0.3729	0.3764	0.3816	0.3862	0.3918	0.3980	0.4034	0.4084	0.4338
MQ2008											
LifeRank	**0.3698**$^{\blacktriangle}$	**0.3857**$^{\blacktriangle}$	**0.4085**$^{\blacktriangle}$	**0.3976**$^{\blacktriangle}$	**0.4501**$^{\blacktriangle}$	**0.4659**$^{\blacktriangle}$	**0.4791**$^{\blacktriangle}$	**0.4859**$^{\blacktriangle}$	**0.4908**$^{\blacktriangle}$	**0.4970**$^{\blacktriangle}$	**0.4686**$^{\blacktriangle}$
LifeRank^{NO}	0.3295	0.3519	0.3689	0.3934	0.4144	0.4337	0.4446	0.4550	0.4615	0.4682	0.4346
OHSUMED											
LifeRank	**0.5330**	**0.4866**	**0.4680**$^{\triangle}$	**0.4681**$^{\blacktriangle}$	**0.4601**$^{\blacktriangle}$	**0.4494**$^{\blacktriangle}$	**0.4436**$^{\blacktriangle}$	**0.4423**$^{\blacktriangle}$	**0.4370**$^{\blacktriangle}$	**0.4366**$^{\blacktriangle}$	**0.4358**$^{\blacktriangle}$
LifeRank^{NO}	0.4571	0.4383	0.4111	0.4014	0.3915	0.3844	0.3784	0.3710	0.3689	0.3632	0.3963

analysed. Moreover, nonlinear feature extraction techniques like some kernel tricks could be incorporated in LifeRank to further improve its performance. Besides, we plan to try more learning to rank loss functions like some state-of-the-art listwise loss functions for performance gains of our algorithm. In addition, we believe it would be interesting to establish theoretical results on dimension reduction for ranking, including feature extraction and feature selection-based algorithms, especially concerning retrieval performance.

Acknowledgements We would like to thank our anonymous reviewers for valuable comments and suggestions.

This research was supported by Ahold Delhaize, Amsterdam Data Science, the Bloomberg Research Grant program, the Dutch national program COMMIT, Elsevier, the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 312827 (VOX-Pol), the Google Faculty Research Award program, the Microsoft Research Ph.D. program, the Netherlands Institute for Sound and Vision, the Netherlands Organization for Scientific Research (NWO) under project nrs CI-14-25, 652.-002.001, 612.001.551, 652.001.003, and Yandex.

All content represents the opinion of the author(s), which is not necessarily shared or endorsed by their respective employers and/or sponsors.

References

1. Arfken GB (2013) Mathematical Methods for Physicists. Academic Press
2. Bach FR, Jordan MI (2005) A probabilistic interpretation of canonical correlation analysis. Tech. Rep. 688, Department of Statistics, University of California, Berkeley
3. Baeza-Yates R, Ribeiro-Neto B (1999) Modern Information Retrieval. Addison Wesley
4. Bertsekas DP (1999) Nonlinear Programming. Athena Scientific
5. Blum AL, Langley P (1997) Selection of relevant features and examples in machine learning. Artif Intell 97(1):245–271
6. Burges C, Shaked T, Renshaw E, Lazier A, Deeds M, Hamilton N, Hullender G (2005) Learning to rank using gradient descent. In: ICML, pp 89–96
7. Burges CJ, Ragno R, Le QV (2007) Learning to rank with nonsmooth cost functions. In: NIPS, pp 193–200
8. Busa-Fekete R, Kégl B, Éltetö T, Szarvas G (2013) Tune and mix: Learning to rank using ensembles of calibrated multi-class classifiers. Machine Learning 93(2-3):261–292
9. Cao Y, Xu J, Liu TY, Li H, Huang Y, Hon HW (2006) Adapting ranking SVM to document retrieval. In: SIGIR, pp 186–193
10. Cao Z, Qin T, Liu TY, Tsai MF, Li H (2007) Learning to rank: From pairwise approach to listwise approach. In: ICML, pp 129–136
11. Chapelle O, Chang Y, Liu TY (2011) Future directions in learning to rank. Journal of Machine Learning Research 14:91–100
12. Chen W, Liu TY, Lan Y, Ma ZM, Li H (2009) Ranking measures and loss functions in learning to rank. In: NIPS, pp 315–323
13. Cossock D, Zhang T (2008) Statistical analysis of Bayes optimal subset ranking. IEEE Transactions on Information Theory 54(11):5140–5154
14. Crammer K, Singer Y (2001) Pranking with ranking. In: NIPS, pp 641–647
15. Freund Y, Iyer R, Schapire RE, Singer Y (2003) An efficient boosting algorithm for combining preferences. Journal of Machine Learning Research 4(1):933–969
16. Geng X, Liu T, Qin T, Li H (2007) Feature selection for ranking. In: SIGIR, pp 407–414
17. Gupta P, Rosso P (2012) Expected divergence based feature selection for learning to rank. In: COLING, pp 431–440
18. Hardoon DR, Szedmak S, Shawe-Taylor J (2004) Canonical correlation analysis: An overview with application to learning methods. Neural Computation 16(12):2639–2664
19. Järvelin K, Kekäläinen J (2002) Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems 20(4):422–446
20. Joachims T, Li H, Liu TY, Zhai C (2007) Learning to rank for information retrieval (LR4IR 2007). SIGIR Forum 41(2):58–62
21. Joachims T, Finley T, Yu CNJ (2009) Cutting-plane training of structural SVMs. Machine Learning 77(1):27–59
22. Joachims T, Swaminathan A, de Rijke M (2018) Deep learning with logged bandit feedback. In: ICLR 2018

23. Jolliffe I (2002) Principal Component Analysis. Springer Series in Statistics, Springer
24. Kendall MG (1948) Rank Correlation Methods. C. Griffin
25. Koren Y, Bell R, Volinsky C (2009) Matrix factorization techniques for recommender systems. Computer 42(8):30–37
26. Lai H, Pan Y, Tang Y, Yu R (2013) FSMRank: Feature selection algorithm for learning to rank. IEEE Transactions on Neural Networks and Learning Systems 24(6):940–952
27. Lan Y, Guo J, Cheng X, Liu TY (2012) Statistical consistency of ranking methods in a rank-differentiable probability space. In: NIPS, pp 1232–1240
28. Lange K (2010) Singular value decomposition. In: Numerical Analysis for Statisticians, Springer, pp 129–142
29. Laporte L, Flamary R, Canu S, Déjean S, Mothe J (2014) Nonconvex regularizations for feature selection in ranking with sparse SVM. IEEE Transactions on Neural Networks and Learning Systems 25(6):1118–1130
30. Lawson C, Hanson R (1995) Solving Least Square Problems, Classics in Applied Mathematics, vol 15. SIAM, Philadelphia
31. Li P, Wu Q, Burges CJ (2007) McRank: Learning to rank using multiple classification and gradient boosting. In: NIPS, pp 897–904
32. Liu TY (2009) Learning to rank for information retrieval. Foundations and Trends in Information Retrieval 3(3):225–331
33. Liu TY (2011) Learning to Rank for Information Retrieval. Springer Science & Business Media
34. Metzler DA (2007) Automatic feature selection in the markov random field model for information retrieval. In: CIKM, ACM, pp 253–262
35. Motoda H, Liu H (2002) Feature selection, extraction and construction. Communication of IICM (Institute of Information and Computing Machinery, Taiwan) 5:67–72
36. Mukuta Y, Harada T (2014) Probabilistic partial canonical correlation analysis. In: ICML, pp 1449–1457
37. Naini KD, Altingövde IS (2014) Exploiting result diversification methods for feature selection in learning to rank. In: ECIR, Springer, pp 455–461
38. Ng AY (2004) Feature selection, L1 vs. L2 regularization, and rotational invariance. In: ICML, pp 78–82
39. Niu S, Guo J, Lan Y, Cheng X (2012) Top-K learning to rank: Labeling, ranking and evaluation. In: SIGIR, pp 751–760
40. Pan F, Converse T, Ahn D, Salvetti F, Donato G (2009) Feature selection for ranking using boosted trees. In: CIKM, ACM, pp 2025–2028
41. Platt JC, Barr AH (1988) Constrained differential optimization for neural networks. Tech. Rep. TR-88-17, Department of Computer Science, California Institute of Technology
42. Qin T, Liu TY (2013) Introducing LETOR 4.0 datasets. arXiv 1306.2597
43. Qin T, Liu TY, Xu J, Li H (2010) LETOR: A benchmark collection for research on learning to rank for information retrieval. Information Retrieval 13(4):346–374

44. Roweis ST, Saul LK (2000) Nonlinear dimensionality reduction by locally linear embedding. Science 290:2323–2326
45. Schölkopf B, Smola A, Müller KR (1998) Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation 10(5):1299–1319
46. Schuth A, Oosterhuis H, Whiteson S, de Rijke M (2016) Multileave gradient descent for fast online learning to rank. In: WSDM 2016: The 9th International Conference on Web Search and Data Mining, ACM, pp 457–466
47. Severyn A, Moschitti A (2015) Learning to rank short text pairs with convolutional deep neural networks. In: SIGIR, ACM, pp 373–382
48. Shalit U, Chechik G (2014) Coordinate-descent for learning orthogonal matrices through Givens rotations. In: ICML, pp 548–556
49. Shivanna R, Bhattacharyya C (2014) Learning on graphs using orthonormal representation is statistically consistent. In: NIPS, pp 3635–3643
50. Tenenbaum JB, de Silva V, Langford JC (2000) A global geometric framework for nonlinear dimensionality reduction. Science 290(5500):2319–2323
51. Tipping ME, Bishop CM (1999) Probabilistic principal component analysis. Journal of the Royal Statistical Society, Series B 61(3):611–622
52. Tsai MF, Liu TY, Qin T, Chen HH, Ma WY (2007) FRank: A ranking method with fidelity loss. In: SIGIR, ACM, pp 383–390
53. Valizadegan H, Jin R, Zhang R, Mao J (2009) Learning to rank by optimizing NDCG measure. In: NIPS, pp 1883–1891
54. Volkovs M, Zemel RS (2009) Boltzrank: Learning to maximize expected ranking gain. In: ICML, pp 1089–1096
55. Wang S, Wu Y, Gao BJ, Wang K, Lauw HW, Ma J (2015) A cooperative coevolution framework for parallel learning to rank. IEEE Transactions on Knowledge and Data Engineering 27(12):3152–3165
56. Weston J, Mukherjee S, Chapelle O, Pontil M, Poggio T, Vapnik V (2000) Feature selection for SVMs. In: NIPS, pp 668–674
57. Wolf L, Bileschi S (2005) Combining variable selection with dimensionality reduction. In: CVPR, pp 801–806
58. Wyse N, Dubes R, Jain A (1980) A critical evaluation of intrinsic dimensionality algorithms. In: Gelsema E, Kanal L (eds) Pattern Recognition in Practice. Proc. workshop Amsterdam, May 1980, North-Holland, pp 415–425
59. Xu J, Li H (2007) AdaRank: A boosting algorithm for information retrieval. In: SIGIR, ACM, pp 391–398
60. Yu H, Oh J, Han W (2009) Efficient feature weighting methods for ranking. In: CIKM, ACM, pp 1157–1166
61. Yue Y, Finley T, Radlinski F, Joachims T (2007) A support vector method for optimizing average precision. In: SIGIR, ACM, pp 271–278

PII

VECTORS OF PAIRWISE ITEM PREFERENCES

by

Gaurav Pandey, Shuaiqiang Wang, Zhaochun Ren, Yi Chang 2019

European Conference on Information Retrieval

https://doi.org/10.1007/978-3-030-15712-8_21

Vectors of Pairwise Item Preferences

Gaurav Pandey[1], Shuaiqiang Wang[2], Zhaochun Ren[2] and Yi Chang[3]

[1] University of Jyvaskyla, Finland
gaurav.g.pandey@jyu.fi
[2] JD.com, China
wangshuaiqiang1,renzhaochun@jd.com
[3] Jilin University, China yichang@ieee.org

Abstract. Neural embedding has been widely applied as an effective category of vectorization methods in real-world recommender systems. However, its exploration of users' explicit feedback on items, to create good quality user and item vectors is still limited. Existing neural embedding methods only consider the items that are accessed by the users, but neglect the scenario when a user gives high or low rating to a particular item. In this paper, we propose *Pref2Vec*, a method to generate vector representations of pairwise item preferences, users and items, which can be directly utilized for machine learning tasks. Specifically, *Pref2Vec* considers users' pairwise item preferences as elementary units. It vectorizes users' pairwise preferences by maximizing the likelihood estimation of the conditional probability of each pairwise item preference given another one. With the pairwise preference matrix and the generated preference vectors, the vectors of users are yielded by minimizing the difference between users' observed preferences and the product of the user and preference vectors. Similarly, the vectorization of items can be achieved with the user-item rating matrix and the users vectors. We conducted extensive experiments on three benchmark datasets to assess the quality of item vectors and the initialization independence of the user and item vectors. The utility of our vectorization results is shown by the recommendation performance achieved using them. Our experimental results show significant improvement over state-of-the-art baselines.

Keywords: Vectorization, Neural Embedding, Recommender systems

1 Introduction

Based on neural networks, neural embedding has emerged as a successful category of vectorization techniques in recommender systems [8, 2], among which *word2vec* [22, 23] is a fundamental and effective algorithm. It was initially proposed for natural language processing problems and considers two states 1 or 0 for each word, representing either appearance or absence of the word in documents. It assumes that the words appearing closer to each other would have higher statistical dependence. Given its effectiveness, many variants have been proposed for machine learning problems, such as name speech recognition [25],

entity resolution [19], machine translation [30], social embedding [12, 24] and recommender systems [11, 1]. Several pioneering efforts have been applied to real-world recommendation scenarios with neural embedding like *prod2vec* [11] and *item2vec* [1], that have been proposed by straightforwardly employing *word2vec*, where each user is considered as a document, and each item is simply regarded as a word. Consequently each item can only have two possible states 1 or 0, representing whether the user has performed a particular action (e.g. purchase, click, etc.) on the item or not. Using sets and sequences of items for each user, they learn the vector representations of the items.

Though such representations create good quality item vectors for some tasks, they lack the functionality to capture higher levels of granularities of users' feedback for vectorization. This could lead to incorrect interpretations, as the top-ranked item and low-ranked items would be treated equally. Thus it is expected to severely limit the vectorization quality for many tasks like calculating item similarities for single item recommendations, clustering user or items, etc. Currently, the efforts are limited for neural embedding-based methods, especially for datasets involving ratings. Therefore, we investigate the neural item embedding problem, to create quality vectorization for items using users' historical rating information with higher granularities (e.g. ratings in range 1 to 5).

To solve this problem, we propose *Pref2Vec* which involves three components: (1) The first step transforms the given user-item rating matrix into a users' pairwise preference matrix. On doing this, each pairwise preference of items has one of the two statuses i.e. occurrence or absence, which is similar to the situation of words in *word2vec*. (2) Then we employ neural embedding to create vector representations for pairwise item preferences by maximizing the likelihood estimation of the conditional probability of each pairwise item preference given another one. Using these preference vectors, the vectors of users can be generated by minimizing the difference between users' observed preferences and the product of the user and preference vectors in the second step of *Pref2Vec*. (3) In the last step, using the user vectors, the item vectors are generated similarly by minimizing the difference between items' observed ratings and the product of user and item vectors.

We evaluate the effectiveness of our *Pref2Vec* method in three experimental tasks on movie recommendation datasets to demonstrate its promising performance, where items are the movies for which user ratings are provided. (1) In the first task, we assess the quality of item vectors, by considering the movie genres as ground-truths. We find the similarities between each pair of items, using the generated item vectors and then using the ground truth (genres). The difference between these two similarities for item pairs are considered as the errors, using which we are able to compute RMSE (root mean squared error) and MAE (mean absolute error), as a quality measures for comparison. We contrast the quality of our item vectors with the quality of item vectors of other standard techniques, like: a) item vectors generated using matrix factorization and b) neural embedding item vectorization by using the sets of items that are rated by users as words. (2) In the second task, we run the vectorizations of the user and

item vectors multiple times. We calculate the average variance of the generated values and the mean average covariance of the generated vectors, to establish that our vectorization process is highly independent of initialization. We contrast this with the vectorizations generated by matrix factorization. (3) Moreover, we compare the recommendation ranking generated using *Pref2Vec* with the standard collaborative filtering algorithms using the NDCG measure. Our results for these experimental tasks show performance gains over the comparison partners.

2 Related Work

Vectorization techniques are of great importance in machine learning. Specially in the area of natural language processing, neural embedding techniques for vectorization of words have been used in many applications [27, 29, 2, 8, 28, 30, 25]. Neural embedding techniques assume that the words that occur close to each other in the text are more dependent than the words that are far off. However, vectorization techniques using neural networks were inefficient to train, especially when the size and vocabulary of the dataset increased. But, the widely used word embedding technique *word2vec* that was introduced a few years ago, made creation of vector representations of words very efficient. It employs highly scalable skip-gram language model, that is fast to train and preserves the semantic relationships of the words in their vector representations. This technique for word embedding has recently shown considerable improvement in applications like name entity resolution [19] and word sense detection [3].

The success of *word2vec* has probably lead to the adoption of the neural embedding techniques in domains other than word representations. Djuric et al. [9] used vectorization of paragraphs as well as vectorization of words contained in each paragraph to create a hierarchical neural embedding framework. Also, Le et al. [20] created an algorithm that learns vector representations of sentences and text documents. They represent each document as dense vector that is utilized to predict words in the document. Moreover, Bordes et al. [4] have introduced the approach that embeds entities and relationships of multi-relational data in low-dimensional vector spaces, to be used for text classification and sentiment analysis tasks. Socher et al. [26] attempted to improve this approach by representing entities as an average of their constituting word vectors. Also, there have been recent efforts to learn the vector representations of nodes in graphs [24, 12].

Moreover, several recent recommendation applications have employed neural word embedding. of *prod2vec* and *user2vec* by Grbovic et al. [11]. The *prod2vec* model creates vector representations of products by employing neural embedding on sequences of product purchases, where each product purchase is considered as a word. Whereas, the *user2vec* model considers a user as a global context in order to learn the vector representations of user and products. Similarly, *item2vec* [1] employs neural embedding on sets of items on which the user has taken action (e.g. songs played or products purchased), while ignoring the sequential information. The experimental results for these techniques show their effectiveness.

Although there have been many applications of neural embeddings in various areas including collaborative filtering, to the best of our knowledge, among the

available neural embedding techniques on the rating information, there is no straightforward way to incorporate different levels of item ratings. He et al. [13] have utilized deep neural network frameworks for recommendation, but they also consider items in 1 and 0 state. Besides, there has not been an attempt to generate and utilize preference vectors. Hence, in this paper we attempt to generate preference vectors as an intermediate step, which can be utilized to generate good quality user and item vectors for various data mining tasks.

3 Problem Formulation

In this section, we formulate the neural rating vectorization problem, aiming to create vector representations for users and items by considering users' historical rating preference on items. Since matrix factorization can be actually regarded as traditional preference vectorization technique, let's firstly review its definition.

Consider a set of users U with m users, a set of items I with n items and a rating matrix R of dimension $m \times n$ containing ratings on n items given by m users. Each element $r_{u,i}$ of the uth row and ith column of R is the rating given by a particular user $u \in U$ for the item $i \in I$, where most of the elements in R are unknown as users generally can provide ratings only for a very small number of items. The objective of the rating vectorization problem is to generate a vector $\boldsymbol{u}$ for each user $u \in U$ and a vector $\boldsymbol{i}$ for each item $i \in I$, where the dot product of each user u and item i is close to the corresponding rating $r_{u,i}$ of i by u. Formally, the problem can be defined as follows:

Definition 1 (Matrix Factorization). *Given a set of users U with m users, a set of items I with n items, a rating matrix R of dimension $m \times n$ containing ratings on n items given by m users, the matrix factorization problem aims to create two low-rank dimensional matrices $\boldsymbol{U}$ of dimension $k \times m$ and $\boldsymbol{V}$ of dimension $k \times n$ for users and items respectively by minimizing the following objective function:*

$$\arg\min_{\boldsymbol{U},V} \sum_{u \in U, i \in I} \phi_{u,i} \left(r_{u,i} - \boldsymbol{u}^\top \boldsymbol{i} \right),$$

where $\phi_{u,i} = 1$, if u has rated i; otherwise 0, We define a novel neural rating vectorization problem. It treats the possible ratings on each item i as an intrinsic property of the item, which indicates the quality of i and thus are independent from users. The neural rating vectorization problem aims to generate rating vectors on items by maximizing the likelihood estimation of the conditional probability of each score on item given another one. Formally, the neural item embedding problem can be defined as:

Definition 2 (Neural Item Embedding). *Let U, I and R be a set of users U with m users, a set of items I with n items, and a rating matrix R of dimension $m \times n$ containing ratings on n items given by m users, respectively. The neural item embedding problem aims to create low rank vector representations of*

dimension $k \times m$ for items I by minimizing the following objective function:

$$\arg\min_{\boldsymbol{I}} - \sum_{u \in U} \sum_{i,j \in I, i \neq j} \phi_{u,i} \phi_{u,j} \log Prob(r_i = r_{u,i} \mid r_j = r_{u,j}), \tag{1}$$

where $Prob(r_i = r_{u,i} \mid r_j = r_{u,j})$ is the probability that user u provides a score of $r_{u,i}$ to item i given that the same user u assigns a score of $r_{u,j}$ to another item j. Once we obtain the item vectors by solving the above problem, user vectors can be generated directly by minimizing the difference between items' observed ratings and the product of user and item vectors: $\arg\min_{\boldsymbol{U}} \sum_{u \in U, i \in I} \phi_{u,i} \left(r_{u,i} - \boldsymbol{u}^\top \boldsymbol{i}\right)$

Note that the probability $Prob(r_i = r_{u,i} \mid r_j = r_{u,j})$ in Equation (1) actually involves two aspects of information: (1) the co-occurrence of ratings on each pair of items by same users, and (2) the rating scores or relative preferences of users holds on items. Thus it is extremely hard to be formulated by straightforwardly adapting that in *word2vec* [22, 23] with hierarchical softmax of the vectors.

4 The *Pref2Vec* Algorithm

Pref2Vec solves the neural item embedding problem in Definition 2 in three steps. Firstly, we generate vectors of pairwise item preference. We use these preference vectors in the second step to generate user vectors, that are in turn used to create item vectors in the third step.

4.1 Pairwise Preference Vectorization

To create vectors of pairwise item preferences, we create the pairwise preference matrix and use it to create the sets of positive pairwise preferences for each user. Then, we utilize neural language models to learn representations of positive preferences in lower dimensional space using available positive preference pairs.

Consider a set of users $U = \{u_1, u_2, \ldots, u_m\}$, a set of items $I = \{I_1, I_2, \ldots, I_n\}$ and their corresponding rating matrix R of dimension $m \times n$. Each row of R contains ratings $R_u = \{r_1, r_2, \ldots, r_n\}$ given by a user u for the n items, where most of the elements in R_u are unknown as users generally can provide ratings only for a very small number of items. This allows us to build a set of pairwise preference for each user by using a preference function: $p(i, j) \in \{+1, -1\}$, where $i = 1 \ldots n$, $j = 1 \ldots n$, $i \neq j$ and both r_i and r_j are known. The preference function $p(i, j)$ has a value of $+1$ if $r_i > r_j$ and -1 otherwise.

Now, we create the sets of positive preferences P_u for each user u. Without losing generality, here we only consider the conditions of positive preference pairs, as all of the negative preferences can be straightforwardly transformed into positive ones by reversing the positions of the two items. With n items, we should consider a total of $N = n(n-1)$ unique preference pairs, denoted as $P = \{p_1, p_2, \ldots, p_N\}$. Each users' preferences P_u is a subset of P, formally $P_u \subseteq P$ for any user u. Now, *Pref2Vec* proceeds with learning the vector representations of the preferences on the collection of preference sets $\mathbb{P} = \{P_1, P_2, \ldots, P_m\}$ for all of the users.

We consider the *word2vec* framework [22, 23] that generates vector representations of words. They presented the continuous *skip-gram* model, which assumed that for each target word the sequence of its surrounding words are trivial and can be ignored. This is achieved by maximizing the cumulative logarithm of the conditional probability for the surrounding words given each target word in the corpus with neural networks. Our approach is very similar, since we consider our collection of preference sets: $\mathbb{P}$ as the corpus, the preference sets $P_1, P_2, \ldots, P_m$ by the users as the sentences and the preferences $p_1, p_2, \ldots, p_N$ as the words.

However, the key difference in our approach is that we completely ignore the spatial information within the preference sets. This is because unlike words in sentences, the order of the preferences for a user (in a non-temporal setup) is inconsequential. This is the reason why we have a set representation of preferences for a user, as opposed to a sequence representation. Actually this property makes our scenario even better fit the *skip-gram* model than natural language processing, where the preferences have no sequence information and thus the sequence of the "surrounding preferences" can be ignored without any accuracy loss. Therefore, in the *Pref2Vec* framework, we learn the vector representations of the products by minimizing the following objective function over the entire collection $\mathbb{P}$ of preference sets:

$$\underset{\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_n}{\arg\min} - \sum_{P_k \in \mathbb{P}} \sum_{(p_i, p_j) \in P_k, i \neq j} \log Prob(p_j \mid p_i), \tag{2}$$

where $Prob(p_j \mid p_i)$ is the hierarchical softmax of the respective vectors of the preference p_j and p_i. In particular, $Prob(p_j \mid p_i) = \frac{\exp(\boldsymbol{i}_o^\top \boldsymbol{j}_t)}{\sum_{p_l \in P_k} \exp(\boldsymbol{i}_o^\top \boldsymbol{l}_t)}$,where $\boldsymbol{i}_o$ and $\boldsymbol{j}_t$ are the initial and target vector representations respectively of preferences p_i and p_j. $\boldsymbol{l}_t$ is the target vector representations of any preference p_l in P_k. From Equation (2), we see that *Pref2Vec* model ignores the sequence of preferences within a user preferences set. The context is set to the level of preference sets, where the preference vectors that fall in the same preference sets will have similar vector representations.

Remarks: Our approach is also inspired by *item2vec* [1], that uses a straightforward application of *word2vec* by considering a set of items (accessed by a user) as a sentence and the individual items as words. Similar to *Pref2Vec*, *item2vec* also ignores the sequential information of items in a set. *item2vec* has been efficiently used in scenarios where we have a simple sequence of items, e.g. products purchased, videos watched, etc. In such cases, for each user the items are in 0 or 1 state. However, if the user feedback is provided in higher granularities (e.g. user ratings), then simply considering the sequence of items rated by the user and treating them equally, is expected to severely limit the quality of vectors. On the other hand, *Pref2Vec* enables the utilization of rating information by incorporating pairwise item preferences in the vectorization process.

4.2 User Vector Generation

However, the preference vectors generated in the previous section cannot be utilized directly for recommendation tasks, that often require good quality user and

item vectors as an input. In this section we describe the second step of *Pref2Vec* and aim to find vectors corresponding to the m users, given the preference vectors for each pair of items and known ground truths for the preferences.

For a particular user let $\boldsymbol{p}_1, \boldsymbol{p}_2 \dots \boldsymbol{p}_r$ be the preference vectors, each of length k, for which the respective values of preference function are $p_1, p_2, \dots, p_r \in \{+1, -1\}$. The corresponding user vector can be achieved by minimizing the cumulative difference between users' each observed preference p_i and the product of the user and preference vectors $\boldsymbol{u}^\top \boldsymbol{p}_i$. Thus we can formulate this problem as linear classification, where $\boldsymbol{p}_1, \boldsymbol{p}_2 \dots \boldsymbol{p}_r$ are training instances, the values of the preference functions $p_1, p_2, \dots, p_r$ are ground truth. With consideration of a bias b, we aim to predict the coefficients of a linear classification model, which is the user vector $\boldsymbol{u}$. In this study, we use Logistic regression [16] to solve the problem. The loss function with L2 norm is : $\arg\min_{\boldsymbol{u},b} \sum_{i=1}^{r} \log(1 + \exp(-p_i(\boldsymbol{u}^T \boldsymbol{p}_i + b))) + \frac{\lambda}{2}||\boldsymbol{u}||^2$, where $\boldsymbol{u}$ is a vector of length k, b is a number and λ is the tuning parameter for L2 norm. We use the gradient descent method for optimization. Given a learning rate α, the update formulas are derived as follows:

$$\begin{aligned} \boldsymbol{u} &\leftarrow \boldsymbol{u} - \alpha\left(\sum_{i=1}^{r} \frac{-p_i}{1 + \exp(p_i(\boldsymbol{u}^T \boldsymbol{p}_i + b))} \boldsymbol{p}_i + \lambda \boldsymbol{u}\right) \\ b &\leftarrow b - \alpha\left(\sum_{i=1}^{r} \frac{-p_i}{1 + \exp(p_i(\boldsymbol{u}^T \boldsymbol{p}_i + b))}\right) \end{aligned} \tag{3}$$

The generated user vectors $\boldsymbol{u}$ corresponding to each of the m users, form a user matrix U of dimension $m \times k$.

4.3 Item Vectors Generation

The last step of *Pref2Vec* is to find item vectors given the rating matrix $R_{m \times n}$ and the user matrix $U_{m \times k}$ generated in the previous section. For this we optimize matrix $I_{n \times k}$, by minimizing the difference between items' observed ratings and the product of user and item vectors, i.e. $UI^\top \approx R$. The n rows of I would be the item vectors. We minimize the loss function: $\arg\min_I ||R - UI^\top||^2 + \frac{\lambda}{2}||I||^2$, where λ is the tuning parameter for L2 normalization. We use the gradient descent method for optimization. Given a learning rate η, the update formula is:

$$I \leftarrow I - \eta(-2(R - UI^T)^\top U + \lambda I) \tag{4}$$

5 Experiments

The following three research questions guide the remainder of the paper.
RQ1 Is the quality of item vectors generated using the *Pref2Vec* approach better than state-of-the-art vectorization algorithms? (See Section 5.1)
RQ2 Are the outputs of the proposed *Pref2Vec* algorithms independent from their initialization? (See Section 5.2)

RQ3 Can the vectorization results be utilized to improve the performance of recommender systems? (See Section 5.3)

Datasets. We use three MovieLens[4] data sets in our experiments: MovieLens-100K, MovieLens-1M and MovieLens-10M. MovieLens-100K dataset contains 100,000 ratings given by 943 users on 1682 movies. MovieLens-1M dataset is larger with 1,000,000 ratings given by 6040 users on 3952 movies. Movielens-10M is the largest dataset used, with 10 million ratings given by 69878 users on 10681 movies. In MovieLens-100K as well as MovieLens-1M the ratings are given as integers from 1 to 5. In MovieLens-10M, the ratings are given in the range 0.5 to 5 with an increment of 0.5. In these datatsets there are 18 movie genres, a movie can belong to one or more of them. For all the three datasets we randomly assign 10 ratings for each user for testing and the rest for training. We have used the vector length of 10 for all the vectorization methods.

5.1 Evaluation of Item Quality

Ground-truth. Since the datasets provide genre information for all of the items (movies), we use the genre similarity as the ground truth. In particular, the genres of each movie are provided (or can be transformed) in the form of binary values. A value of 1 signifies that the movie belongs to a particular genre and 0 signifies the contrary. A movie can belong to more than one genre. So, let us consider that genre vectors derived from the meta-data are: $(\boldsymbol{G}_i \dots \boldsymbol{G}_j)$, which correspond to our item vectors $\boldsymbol{I}_i \dots \boldsymbol{I}_j$. Since, the genre vectors are binary vectors, to find similarity between them we use: Jaccard similarity [6], an efficient and popular measure for binary similarity. Jaccard similarity between two binary vectors $\boldsymbol{v}_a$ and $\boldsymbol{v}_b$ is simply calculated as: $jacSim(\boldsymbol{v}_a, \boldsymbol{v}_b) = \frac{F_{11}}{F_{01}+F_{10}+F_{11}}$,where F_{11} is the number of features for which both $\boldsymbol{v}_a$ and $\boldsymbol{v}_b$ have value 1. F_{01} is the number of features for which $\boldsymbol{v}_a$ has value 0 and $\boldsymbol{v}_b$ has 1. And, F_{10} is the number of features where $\boldsymbol{v}_a$ had the value 1 and $\boldsymbol{v}_b$ has 0.

For the item vectors $\boldsymbol{I}_1, \boldsymbol{I}_2, \dots, \boldsymbol{I}_n$ (calculated in Section 4.3), the similarity can be calculated for each pair of item vectors $(\boldsymbol{I}_i, \boldsymbol{I}_j)$ as: $cosSim(\boldsymbol{I}_i, \boldsymbol{I}_j) = \frac{\boldsymbol{I}_i^\top \boldsymbol{I}_j}{|\boldsymbol{I}_i| \times |\boldsymbol{I}_j|}$, where $|\boldsymbol{I}_i|$ and $|\boldsymbol{I}_i|$ are the length of the vectors $\boldsymbol{I}_i$ and $\boldsymbol{I}_j$.

Evaluation Metrics. In order to evaluate the quality of item vectors we use the RMSE (root mean squared error) and MAE (mean absolute error) measures. To calculate these, we calculate the similarities between each pair of item vectors and the similarities between their corresponding pairs of ground truths. Since the item in our experiments are movies, the genre information about the movies (available from metadata) is considered as ground truth. The differences between the two similarities for each item are considered as errors, that are in turn used to calculate RMSE and MAE. We use these measures because for good quality item vectors, the vectors that are similar should also have similarity based on their relevant meta data information. Therefore, the lower the values of RMSE and MAE, the better is the quality of vectors.

[4] http://grouplens.org/datasets/movielens/

Table 1. Quality of Generated Item Vectors against Baselines

Algorithm	ML-100K		ML-1M		ML-10M		Ref.
	RMSE	MAE	RMSE	MAE	RMSE	MAE	
RM-Vectors	0.8674	0.8163	0.8790	0.8292	0.8563	0.8151	-
IS-Vectors	0.8238	0.7508	0.7018	0.5886	0.5864	0.4758	[1]
MF-Vectors	0.6844	0.6431	0.6904	0.6478	0.6478	0.6071	[18]
P2V-Vectors	**0.4770**	**0.3846**	**0.5165**	**0.4305**	**0.4456**	**0.3695**	This paper

To calculate the errors we need: the difference between the similarities of two item vectors and the similarities between the corresponding two genre vectors. The errors are calculated for all pairs of items: $e_{i,j} = cosSim(\boldsymbol{I}_i, \boldsymbol{I}_j) - jacSim(\boldsymbol{G}_i, \boldsymbol{G}_j)$. Though cosine similarity and Jaccard similarity are different measurements, their difference used here is expected to be highly indicative of the error. There would be $n(n-1)/2$ such errors. Now, $RMSE = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n} e_{i,j}^2}{n(n-1)/2}}$ and $MAE = \frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n} |e_{i,j}|}{n(n-1)/2}$, where the function $|\cdot|$ gives the absolute value of the parameter.

Baselines. We choose the following methods to evaluate the quality of the item vectors that are generated by the *Pref2Vec* framework, i.e. P2V-Vectors.

• **RM-Vectors:** Rating matrix $R_{m\times n}$ contains ratings by m users for n items, and its columns are the simplest (and readily available) form of item vectors.

• **IS-Vectors:** Neural embeddings of items are created by considering the set of items rated by users as sentences and items as words (similar to *item2vec* [1] approach). Comparison with this method would validate the importance of using preference information for vectorization in *Pref2Vec*.

• **MF-Vectors:** In matrix factorization [18] user and item vectors are created by randomly initializing matrices $U_{m\times k}$ and $I_{n\times k}$ and then minimizing the difference between their product and the rating matrix (i.e. $R - UI^\top$).

Results. In Table 1, we compare the item vector qualities using RMSE and MAE. For MovieLens-100K dataset, for both RMSE and MAE, P2V-Vectors perform the best, followed by MF-Vectors. IS-Vectors are the third and the RM-Vectors are the worst performing ones. The trend is same for the dataset MovieLens-1M for RMSE. For MovieLens-1M in terms of MAE as well as for MovieLens-10M (both RMSE and MAE), though P2V-Vectors are still the best performing ones, the second best are IS-Vectors, followed by MF-Vectors and then RM-Vectors. The improvement shown by P2V-Vectors is significant.

Pref2Vec firstly generates preference vectors, and then creates user vectors with the generated preference vectors, and finally produces item vectors with the generated user vectors. Since each step is an approximation process with certain accuracy loss, the preference and user vectors should be more accurate than the item vectors. Thus although we cannot assess the quality of user and preference vectors resulting from lack of corresponding ground-truth information, we can still claim that the quality of the preference, user and item vectors generated by our *Pref2Vec* method can significantly outperform our baselines.

5.2 Evaluation of Initialization Independence of Generated Vectors

Firstly, we describe the measurements to evaluate the independence of generated vectors from their initialization. Let us consider that x different runs (resulting from different initializations) of a vector generation method generate: user matrices $U^{(1)} \dots U^{(x)}$ and the corresponding item matrices $I^{(1)} \dots I^{(x)}$. Each user matrix is of dimension $m \times k$ with the rows corresponding to m user vectors, each of length k. Similarly each item matrix is of dimension $n \times k$ with the rows corresponding to n item vectors, each of length k. Since the features of the vectorization results might be in a different order by different runs of algorithms, we sort the generated features according to their cumulative values among all of the users. The independence of these vectors from the initialization can be measured using (a) variance of the elements of the U and I matrices and (b) correlations between the user and item vectors generated in different runs. These measures are explained in detail as follows.

Variance Calculation. Let $U^{(1)}_{i,j}, U^{(2)}_{i,j} \dots U^{(x)}_{i,j}$ be the x values in the user matrices at ith row and jth column from x different runs of a vectorization algorithm. Their variances can be calculated as: $var_U(i,j) = \frac{1}{x} \sum_{l=1}^{x} \left(U^{(l)}_{i,j} - \overline{U}_{i,j} \right)^2$, where $\overline{U}_{i,j}$ is the average of $U^{(1)}_{i,j}, U^{(2)}_{i,j} \dots U^{(x)}_{i,j}$. With $m \times k$ dimensions of the user matrix U, we can get $m \times k$ variance, and the mean variance would be: $MVU = \frac{1}{m \times k} \sum_{i=1}^{m} \sum_{j=1}^{k} var_U(i,j)$.

Similarly, the variance of the item matrices at the ith row and jth column $I^{(1)}_{i,j}, I^{(2)}_{i,j} \dots I^{(x)}_{i,j}$ from x different runs of a vectorization algorithm can be calculated as: $var_I(i,j) = \frac{1}{x} \sum_{l=1}^{x} \left(I^{(l)}_{i,j} - \overline{I}_{i,j} \right)^2$, where $\overline{I}_{i,j}$ is the average of $I^{(1)}_{i,j}, I^{(2)}_{i,j}, \dots, I^{(x)}_{i,j}$. The mean variance of the generated item vectors can be calculated as: $MVI = \frac{1}{n \times k} \sum_{i=1}^{n} \sum_{j=1}^{k} var_I(i,j)$

A lower value of the mean variance is indicative that the generated values that comprise the user or item vectors do not vary much with different initializations.

Correlation of Vectors. The independence of the vectors from the initialization of the generation technique can also be estimated by the correlation between the vectors generated in different runs. We use Pearson correlation coefficient [15] to calculate correlation $\rho(x, y)$ between variables x and y.

A user matrix $U^{(j)}$ generated in the j^{th} run, contains m user vectors : $\boldsymbol{u}^{(j)}_1 \dots \boldsymbol{u}^{(j)}_m$. For a particular user, the average of pairwise correlations between the vectors generated in the x runs would be: $AC(i) = \frac{\sum_{j=1}^{x} \sum_{l=j+1}^{x} \rho(\boldsymbol{u}^{(j)}_i, \boldsymbol{u}^{(l)}_i)}{x(x-1)/2}$ And, the mean of these average correlation for all the m user vectors can simply be calculated as: $MAC = \frac{\sum_{i=1}^{m} \sum_{j=1}^{x} \sum_{l=j+1}^{x} \rho(\boldsymbol{u}^{(j)}_i, \boldsymbol{u}^{(l)}_i)}{m \times x(x-1)/2}$

Similarly, the mean average correlation for the item vectors, $\boldsymbol{i}^{(j)}_1 \dots \boldsymbol{i}^{(j)}_n$ generated in x runs ($j = 1 \dots x$), can be calculated as: $MAC = \frac{\sum_{i=1}^{n} \sum_{j=1}^{x} \sum_{l=j+1}^{x} \rho(\boldsymbol{i}^{(j)}_i, \boldsymbol{i}^{(l)}_i)}{n \times x(x-1)/2}$ A high value if MAC mean that the vectors generated during different runs are close to each other and hence have high level of independence to initialization.

Table 2. Initialization Independence of Generated Vectors

Algorithm	User Vectors		Item Vectors	
	MVU	MAC	MVI	MAC
MF	0.0730	0.0015	0.0773	0.0315
P2V	**0.0015**	**0.8592**	**0**	**0.7881**

Results. In Table 2, we show the results evaluating the initialization independence of user and item vectors generated using *Pref2Vec* (shown as P2V) and comparing them with the vectors generated by matrix factorization (shown as MF). On the dataset MovieLens-100K we run both the methods 5 times, resulting in creation of 5 different pairs of user and item vectors for both of them. We calculate MVU, MVI and MAC (for user and item vectors) for the vectorization results generated by P2V and matrix factorization (MF).

The values of MVU and MVI of our algorithm are merely 0.0015 and 0 for user and item vectors, which are sharply lower than that of the matrix factorization method. Note that although the values of matrix factorization are smaller than 0.1, they are still large because the values in the user and item matrices are very small, and most of them are less than 1. Also, the values of MAC are very high for P2V for both item and user vectors, especially in comparison with the respective values for MF. This again shows that the user and item vectors generated by P2V in different runs are highly correlated to each other.

5.3 Ranking Prediction based on Generated Vectors

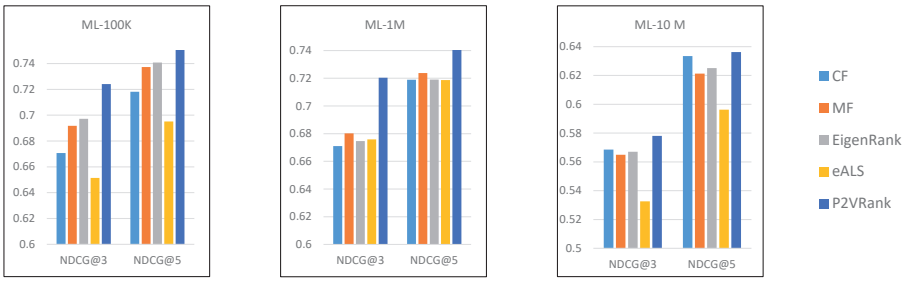
Ranking Model using User Vectors. Here we describe the method to generate rankings for items with unknown ratings for user using the available *Pref2Vec* preference and user vectors. This is done by firstly predicting the preference values $\hat{p} \in \{+1, -1\}$ for the preference vectors corresponding to the items with unknown ratings. Then we employ a greedy order algorithm to derive approximately optimal ranking of the unrated items.

In Section 4.2 we showed the process that generates the user vector $\boldsymbol{u}$ and the value b after optimization. Since the optimization process directly employs the Logistic regression loss function, this allows us to also directly use Logistic regression classification to predict pairwise preferences for a user. More specifically, for a user with user vector $\boldsymbol{u}$ and accompanying value b, the user's preference $\hat{p}_u$ can be predicted as: $\hat{p}_u = +1, \text{if } \boldsymbol{u}^\top \boldsymbol{p} + b > 0; -1$ otherwise.

Hence, for a particular user, if there are q items with unknown rankings $I_1, I_2 \ldots I_q$, the values for the preference function $\hat{p}(I_i, I_j) \in \{+1, -1\}$, can be predicted. Since the values for pairwise preference function are not a direct format to get the rankings, we use the greedy order algorithm proposed by Cohen et al. [7, 21], that efficiently finds an approximately optimal ranking for the target user u. It is showed that based on reduction of cyclic ordering problem [10], the determination of optimal ranking is a NP-complete problem and the algorithm can be proved to have an approximation ratio of 2 [10].

Remarks. Alternatively, we could have directly used the user matrix U (Section 4.2) and the item matrix I (Section 4.3) to generate the ratings matrix ($R =$

Fig. 1. Ranking Performance of *Pref2Vec* against baselines



UI^T), that could be used to generate ranking of unrated items. However, since we follow sequential steps by first generating U from preference vectors, then using U to create I and thereafter using U and I to create R; there is accuracy loss at each step. On the other hand, our ranking model avoids such additional inaccuracies by directly using preference vectors and U to generate rankings.

Baselines. We use the following baselines to access the performance of our simple recommendation method P2VRank:
• **CF:** CF [5] is a memory-based collaborative filtering algorithm that uses the Pearson correlation coefficient to calculate the similarity between users.
• **MF:** Given a raking matrix R, in matrix factorization [18] the user matrix U and the item matrix I are optimized in order to minimize the difference: $R - UI^\top$.
• **EigenRank:** EigenRank [21] uses greedy aggregation method to aggregate the predicted pairwise preferences of items into total ranking.
• **eALS:** Element-wise Alternating Least Squares (eALS) [14] efficiently optimizes a MF model with variably-weighted missing data. As eALS is an implicit feedback algorithm, we consider only higher ratings (≥ 4) as positive feedback.

Results. The performance is evaluated using the standard ranking accuracy metric NDCG [17] @3 and @5. In Fig 1, we see that P2VRank outperforms all comparison partners. Also, we also observed strong statistical significance ($\alpha = 0.05$) on comparing P2VRank against MF for all the three datasets.

6 Conclusion

We proposed *Pref2Vec* to generate vector representations of pairwise item preferences. We also presented the method to generate user and item vectors using preference vectors. Also, our experimental results demonstrated that the quality of item vectors generated by *Pref2Vec* is better than that of the standard techniques. We also verified that the generated user and item vectors are highly independent of the initializations. In addition, we presented the technique to generate rankings of items, using the generated user vectors, and showed that it outperforms the standard recommendation techniques. Currently we only consider the preference of one item over another for the creation of *Pref2Vec* and in future we would like to consider the magnitudes of these preferences.

References

1. Barkan, O., Koenigstein, N.: Item2vec: Neural item embedding for collaborative filtering. In: MLSP. pp. 1–6 (2016)
2. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A neural probabilistic language model. J. Mach. Learn. Res. **3**, 1137–1155 (2003)
3. Bhingardive, S., Singh, D., V, R., Redkar, H.H., Bhattacharyya, P.: Unsupervised most frequent sense detection using word embeddings. In: HLT-NAACL. pp. 1238–1243 (2015)
4. Bordes, A., Usunier, N., Garcia-Durán, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: NIPS. pp. 2787–2795 (2013)
5. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: UAI. pp. 43–52 (1998)
6. Choi, S., Cha, S., Tapper, C.C.: A survey of binary similarity and distance measures. J. Systemics, Cybernetics and Informatics **8**(1), 43–48 (2010)
7. Cohen, W.W., Schapire, R.E., Singer, Y.: Learning to order things. J. Art. Int. Res. **10**(1), 243–270 (1999)
8. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)
9. Djuric, N., Wu, H., Radosavljevic, V., Grbovic, M., Bhamidipati, N.: Hierarchical neural language models for joint representation of streaming documents and their content. In: WWW. pp. 248–255 (2015)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co. (1990)
11. Grbovic, M., Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V., Sharp, D.: E-commerce in your inbox: Product recommendations at scale. In: SIGKDD. pp. 1809–1818 (2015)
12. Grover, A., Leskovec, J.: Node2vec: Scalable feature learning for networks. In: SIGKDD. pp. 855–864 (2016)
13. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: WWW. pp. 173–182 (2017)
14. He, X., Zhang, H., Kan, M.Y., Chua, T.S.: Fast Matrix Factorization for Online Recommendation with Implicit Feedback. In: SIGIR. pp. 549–558 (2016)
15. Herlocker, J., Konstan, J.A., Riedl, J.: An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. ACM Trans. Inf. Syst. **5**, 287–310 (2002)
16. Hosmer Jr, D.W., Lemeshow, S., Sturdivant, R.X.: Applied Logistic Regression, vol. 398. John Wiley & Sons (2013)
17. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst. **20**(4), 422–446 (2002)
18. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
19. Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., Dyer, C.: Neural architectures for named entity recognition. In: HLT-NAACL. pp. 260–270 (2016)
20. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: ICML. pp. 1188–1196 (2014)
21. Liu, N.N., Yang, Q.: Eigenrank: A ranking-oriented approach to collaborative filtering. In: SIGIR. pp. 83–90 (2008)

22. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013)
23. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS. pp. 3111–3119 (2013)
24. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: SIGKDD. pp. 701–710 (2014)
25. Schwenk, H.: Continuous space language models. Computer Speech Lang. **21**(3), 492 – 518 (2007)
26. Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with neural tensor networks for knowledge base completion. In: NIPS. pp. 926–934 (2013)
27. Socher, R., Lin, C.C., Ng, A.Y., Manning, C.D.: Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In: ICML (2011)
28. Turian, J., Ratinov, L., Bengio, Y.: Word representations: A simple and general method for semi-supervised learning. In: ACL. pp. 384–394 (2010)
29. Turney, P.D.: Distributional semantics beyond words: Supervised learning of analogy and paraphrase. TACL **1**, 353–366 (2013)
30. Zou, W.Y., Socher, R., Cer, D.M., Manning, C.D.: Bilingual word embeddings for phrase-based machine translation. In: EMNLP. pp. 1393–1398 (2013)

PIII

RECOMMENDING SERENDIPITOUS ITEMS USING TRANSFER LEARNING

by

Gaurav Pandey, Denis Kotkov, Alexander Semenov 2018

Proceedings of the 27th ACM International Conference on Information and Knowledge Management

Recommending Serendipitous Items using Transfer Learning

Gaurav Pandey
University of Jyvaskyla
gaurav.g.pandey@jyu.fi

Denis Kotkov
University of Jyvaskyla
kotkov.denis.ig@gmail.com

Alexander Semenov
University of Jyvaskyla
alexander.v.semenov@jyu.fi

ABSTRACT

Most recommender algorithms are designed to suggest relevant items, but suggesting these items does not always result in user satisfaction. Therefore, the efforts in recommender systems recently shifted towards serendipity, but generating serendipitous recommendations is difficult due to the lack of training data. To the best of our knowledge, there are many large datasets containing relevance scores (relevance oriented) and only one publicly available dataset containing a relatively small number of serendipity scores (serendipity oriented). This limits the learning capabilities of serendipity oriented algorithms. Therefore, in the absence of any known deep learning algorithms for recommending serendipitous items and the lack of large serendipity oriented datasets, we introduce *SerRec* our novel transfer learning method to recommend serendipitous items. *SerRec* uses transfer learning to firstly train a deep neural network for relevance scores using a large dataset and then tunes it for serendipity scores using a smaller dataset. Our method shows benefits of transfer learning for recommending serendipitous items as well as performance gains over the state-of-the-art serendipity oriented algorithms.

KEYWORDS

Recommender System; Serendipity; Deep Learning; Transfer Learning

ACM Reference Format:
Gaurav Pandey, Denis Kotkov, and Alexander Semenov. 2018. Recommending Serendipitous Items using Transfer Learning. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3269206.3269268

1 INTRODUCTION

Relevance oriented recommender algorithms often suggest items that users are either already familiar with or would easily find themselves leading to low satisfaction [8]. To overcome this problem, recommender algorithms should suggest serendipitous (i.e. relevant, novel and unexpected) items, as these items are more likely to broaden user preferences than relevant non-serendipitous ones [7]. While there has been a lot of work in the area of relevance oriented recommendations including deep learning [1–3, 10], the efforts for serendipitous recommendations are still very limited. To the best of our knowledge, deep learning or transfer learning methods have not yet been explored for recommending serendipitous items.

Although there is abundant availability of big training datasets with relevance scores, that can be used for relevance oriented algorithms, there are not many training datasets available with serendipity scores. To the best of our knowledge, Serepdipity-2018[1] is the only such publicly available dataset, having comparatively a rather small set of serendipity scores. One of the reasons for the lack of large serendipity related datasets is the high level of difficulty to collect user feedback regarding serendipity. This requires a lot of effort from users as they need to answer many questions, and not just provide scores for the items [7]. The unavailability of large datasets with serendipity scores poses limitations on the training of serendipity oriented recommender models.

Therefore, in the absence of large datasets and deep learning algorithms focused on serendipity, we introduce *SerRec*: a transfer learning method that trains a deep neural network for relevance scores using a large dataset and then tunes it for serendipity scores using a smaller dataset. This allows us to use the available training data with relevance scores to benefit in the process of recommending serendipitous items.

Our method utilizes the neural collaborative filtering (NCF) framework proposed by He et al [3]. They introduced an ensemble of deep neural networks, originally proposed to learn a relevance oriented recommender model. To benefit from transfer learning, we firstly train the deep neural network ensemble layers using an available large training dataset with relevance scores. Thereafter, we tune the last layer of the network using a small dataset with serendipity scores.

For our experiments, we have used Serendipity-2018, the only publicly available dataset that has user feedback on serendipitous items [7], that to the best of our knowledge has not been used so far in studies. This dataset consists of a large collection of relevance scores along with a smaller collection of serendipity scores. Experimenting with Serendipity-2018 is also novel in itself, because serendipity oriented algorithms till now have been evaluated on datasets where serendipity was measured using artificial serendipity metrics based on assumptions regarding serendipity that might not correspond to reality [7]. Our experimental results show the benefits of transfer learning to train a serendipity oriented recommender model and shows improvements over the state-of-the-art serendipity oriented recommender models.

To summarize, this paper has the following key contributions:

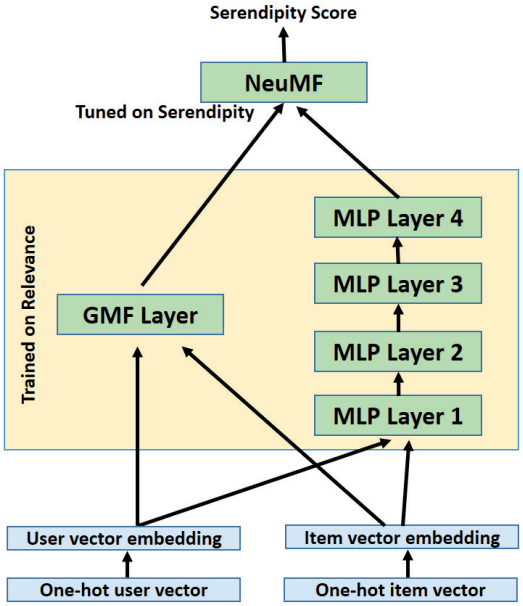
- We propose the novel deep transfer learning method *SerRec* for serendipitous recommendations.
- Our method utilizes the neural collaborative filtering framework to utilize a large relevance score dataset along with a

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3269268

[1] https://grouplens.org/datasets/serendipity-2018/

Figure 1: *SerRec* Architecture for Transfer Learning for Serendipity using NCF framework [3]



smaller serendipity score dataset to enable high performance serendipitous items recommendations.

- We evaluate *SerRec* and compare it with the-state-of-the-art serendipity oriented algorithms on the first publicly available serendipity oriented dataset Serendipity-2018.

2 OBJECTIVES

In this study, we aim to address mainly the following questions:

- How can we do transfer learning for serendipitous recommendations using the relevance score training data?
- Does transfer learning help the serendipity oriented recommender model?
- How does our model compare to the state-of-the-art serendipity oriented models?

3 *SERREC* FRAMEWORK

In this section, we describe the *SerRec* methodology, its setup, utilized datasets, baselines and the metric employed for comparisons.

3.1 Methodology

Consider a set of users as $U = \{u_0, u_1, ..., u_N\}$, and a set of items as $I = \{i_0, i_1, ..., i_M\}$. We denote user-item interaction matrix as $R = \{r_{jk}\}$, where $r_{jk} = 1$ if user u_j rated item i_k, $j \in \{0, ..., N\}, k \in \{0, ..., M\}, r_{jk} \in \{0, 1\}$. We denote user-item serendipity matrix as $S = \{s_{jk}\}$, where $s_{jk} = 1$ if user u_j considers item i_k, where $j \in \{0, ..., N\}, k \in \{0, ..., M\}, s_{jk} \in \{0, 1\}$ as serendipitous.

Our goal is to learn a function $\hat{s}_{jk} = f(j, k \mid \Theta)$, where $\hat{s}_{jk}$ is predicted serendipity score, and Θ is the vector of model parameters. Here, j and k are the indexes of user and item matrices U and I, respectively. In order to find optimal parameters Θ for function $f(j, k \mid \Theta)$ we need to minimize loss function $\mathcal{L}(S, f(j, k \mid \Theta))$ between actual and predicted serendipity scores.

As per He et al. [3], we represent function $f(j, k \mid \Theta))$ as neural network, depicted in Figure 1. Input data for the neural network is one-hot encoded user and item vectors: $\boldsymbol{u}$ and $\boldsymbol{i}$, $|\boldsymbol{u}| = N + 1, |\boldsymbol{i}| = M + 1$. At first, we create vector embeddings $\boldsymbol{p}$ and $\boldsymbol{q}$ from vectors $\boldsymbol{u}$ and $\boldsymbol{i}$ respectively. These embeddings are created by learning weights of two matrices: W_U and W_I; multiplication of $\boldsymbol{u}$ or $\boldsymbol{i}$ to W_U or W_I respectively would return $\boldsymbol{p}$ and $\boldsymbol{q}$ [3]. The sizes of these embeddings $\boldsymbol{p}$ and $\boldsymbol{q}$ are n and m respectively, where $n << N + 1$ and $m << M + 1$.

Left-hand side of the neural network at Figure 1 is the layer performing Generalized Matrix Factorization (GMF), i.e.:

$$O_{GMF} = (\boldsymbol{p} \odot \boldsymbol{q}) \tag{1}$$

Moreover on the right hand side, Multi Layer Perceptron (MLP) layers 1, 2, 3 and 4 contain dense layers:

$$O_{MLP_k} = RelU(W_k \cdot O_{MLP_{k-1}} + B_k), k \in \{1, 2, 3, 4\}, \tag{2}$$

where $O_{MLP_0} = [\boldsymbol{p}, \boldsymbol{q}]$, i.e. vector built by concatenation of user and item embeddings. Final layer of the network Neural Matrix Factorization (NeuMF in Figure 1) implements the sigmoid activation function:

$$O_{NeuMF} = \sigma(W_O \cdot [O_{GMF}, O_{MLP_4}] + B_O), \tag{3}$$

where W_O and B_O are weights and biases of the output layer.

We adopt cross entropy as the loss function:

$$\mathcal{L}(S, \hat{S}) = -\sum_{(j,k)\in\widetilde{S}} (s_{jk} \log \hat{s}_{jk} - (1 - s_{jk}) \log(1 - \hat{s}_{jk})), \tag{4}$$

where $\widetilde{S}$ denotes observed part of serendipity matrix S, and $\hat{S} = f(j, k \mid \Theta)$ is the predicted serendipity.

Since serendipity scores dataset is small, we try to utilize transfer learning by training a deep neural network using relevance scores. First, we train the entire neural network framework on relevance data R, and optimize the loss $\mathcal{L}(R, \hat{R})$. After training the entire neural network on relevance, we fix all weights except final one (W_O, and B_O), and train it on serendipity matrix S.

3.2 *SerRec* Setup

For our experiments we have used the Neural Collaborative Filtering [3] implementation[2] as discussed in section 3.1. As shown in Figure 1, we used the training dataset with relevance scores to train the GMF the MLP layers, where we used four layers in MLP. Thereafter, the NeuMF layer is tuned using the serendipity tuning dataset and final network is tested on the serendipity test dataset. These datasets are explained in detail in Section 3.3.

For initial training on GMF and MLP, we used the user and item vectors embeddings $\mathbf{p}$ and $\mathbf{q}$, created using relevance training dataset $\widetilde{R} \subset R$. We used a learning rate of 0.001 for GMF and 0.01 for MLP, and trained both of them for 20 epochs while keeping a batch size of 254. Also for both GMF and MLP, we used Adam optimization algorithm [5]. Moreover, the available implementations for MLP and GMF convert the available ratings into implicit feedback (rated

[2]https://github.com/hexiangnan/neural_collaborative_filtering

or unrated) and for this chooses randomly four negative samples (unrated) for each positive (rated) item for a user. We have used these default settings.

Then using these trained GMF and MLP layers and keeping the weights fixed in them, we tuned the NeuMF layer using the serendipity tuning dataset. For this we used $\widetilde{S} \subset S$, that contains negative and positive samples on serendipity. We tuned it till convergence using a learning rate of 0.001. We used this trained and tuned network to predict the serendipity scores of the test dataset.

3.3 Datasets

To evaluate our algorithm and baselines, we employed Serendipity-2018 dataset [7]. To the best of our knowledge, this is the only publicly available dataset, which contains user feedback regarding serendipity. This dataset contains 5-star scores (relevance scores) users gave to movies in MovieLens[3] and binary scores (serendipity scores) indicating whether particular movies are serendipitous to particular users. The dataset contains ten million relevance scores and 2,150 serendipity scores.

The dataset contains different kinds of serendipity. In this study, we target six kinds of serendipity that are missing the unexpectedness variation, which hurts user satisfaction: strict serendipity (find), strict serendipity (implicit), strict serendipity (recommend), motivational serendipity (find), motivational serendipity (implicit) and motivational serendipity (recommend) [7]. We pre-process this dataset and regard a movie serendipitous (positive sample) if it is serendipitous according to at least one of these variations of serendipity, and otherwise regard it as non-serendipitous (negative example). The dataset contains 277 serendipitous user movie pairs out of total 2,150.

For the Serendipity-2018 dataset, serendipity scores were obtained in a survey taken by 481 users. In the survey, the authors selected movies that were likely to be serendipitous to users, such as unpopular movies that were given high scores [7]. To extend serendipity scores for our study, we randomly selected five relevance scores per user and assigned negative (non-serendipitous) serendipity scores to them. We regarded these movies non- serendipitous, as they were unlikely to be serendipitous to users. In the best case scenario, the chance of a movie to be serendipitous is 13% ($\frac{277}{2150} = 0.129$). In our case, the chance of a movie to be serendipitous is much lower, since we did not control for popularity or score. After attaching serendipity scores to randomly selected relevance scores, the number of serendipity scores exceeded 4,555 with 277 scores indicating serendipitous movies and 4,278 indicating non-serendipitous ones.

We split the dataset into three datasets: training, tuning and test. Tuning and test datasets contain both relevance and serendipity scores, while the training dataset only contains relevance scores. The tuning datasets contains 75% of serendipity scores, while the test dataset contains the remaining 25% of serendipity scores.

3.4 Baselines

We implemented the following baselines for comparison with our transfer learning method *SerRec*:

- **POP**: We implemented popularity baseline that arranges items according to the number of relevance scores received by them in the training dataset, in the descending order.
- **UNPOP**: Unpopularity or inverse popularity baseline orders items according to the number of relevance scores in the ascending order.
- **Random**: This baseline orders the items randomly.
- **SVD**: Singular value decomposition [6] orders items according to the predicted scores. SVD decomposes the user-item matrix into two matrices using gradient decent. The gradient decent algorithm minimizes the objective function, which is the error between actual and predicted scores. Based on tuning, we picked the parameters: feature number=200, learning rate=10^{-5} and regularization term=0.1.
- **SPR**: Serendipitous Personalized Ranking is a serendipity-oriented variation of SVD with the modified objective function [9]. SPR is a learning to rank algorithm, which maximizes the difference between scores of relevant and irrelevant items for each user and weights this distance based on popularity of the irrelevant item. Based on tuning, we picked the parameters: Bayesian loss function, $\alpha = 0.4$, feature number=200, learning rate=10^{-5} and regularization term=0.1.
- **UAUM**: Unexpectedness-Augmented Utility Model is also a serendipity-oriented variation of SVD [12]. UAUM minimizes the objective function, which is the error weighted with the unexpectedness term. In our implementation, we excluded unobserved scores due to the size of our dataset. Based on tuning, we picked the parameters: feature number=200, learning rate=10^{-5} and regularization term=0.1.
- ***SerRec***$_{NoTL}$: To see if transfer learning indeed helps in recommending serendipitous items, we trained the non transfer learning version of our method, where we trained all the layers in Figure 1 using only serendipity scores from tuning dataset.

We used the following procedure to evaluate our baseline algorithms: (1) we trained the baselines on relevance scores of the training dataset, (2) we tuned the parameters of the baselines on serendipity scores of the tuning dataset, (3) we trained the baselines with the tuned parameters on relevance scores of training and tuning datasets combined and (4) we evaluated the trained baselines on serendipity scores of the test dataset. We did not train the baselines directly on the serendipity scores, as Serendipity-2018 does not contain enough serendipity scores for training the algorithms.

3.5 Metric

To compare the serendipitous item recommendation performance of *SerRec* against the baselines on the serendipity test set, we employed the standard retrieval metric NDCG@1-10 (normalized discounted cumulative gain) [4]. While NDCG computation typically utilizes the relevance scores of items in a ranking, we used the available serendipity scores.

4 RESULTS

Table 1 compares the serendipitous recommendation performance of *SerRec* against the baselines using NDCG@1-10. The following observations can be made from the results:

[3] https://movielens.org/

Table 1: Serendipity Ranking Performance Comparison of *SerRec*

NDCG	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10
POP	0.0303	0.0606	0.0606	0.0909	0.1146	0.1948	0.2518	0.2994	0.3550	0.4078
Random	0.0909	0.1969	0.3262	0.3895	0.4248	0.4529	0.5036	0.5339	0.5639	0.5674
SVD	0.2121	0.3484	0.4035	0.4499	0.4981	0.5298	0.5643	0.6086	0.6218	0.6281
UAUM	0.3333	0.4696	0.4765	0.5117	0.5511	0.5900	0.5900	0.6336	0.6623	0.6749
UNPOP	0.3636	0.4393	0.5017	0.5651	0.6478	0.6633	0.6849	0.6849	0.7040	0.7040
SPR	0.3636	0.5000	0.5678	0.6230	0.6731	0.6944	0.7268	0.7369	0.7369	0.7369
$SerRec_{NoTL}$	0.2727	0.4091	0.5420	0.6490	**0.6960**	0.7135	0.7135	0.7236	0.7236	0.7236
SerRec	**0.4848**	**0.5455**	**0.6269**	**0.6505**	0.6907	**0.7186**	**0.7363**	**0.7452**	**0.7614**	**0.7614**

(1) We see that *SerRec* outperforms all the baselines algorithms at all the NDCG metrics, only except at NDCG@5 where $SerRec_{NoTL}$ is the best performing algorithm.
(2) For some metrics $SerRec_{NoTL}$ is the second best algorithm and SPR the third best, while for other metrics it is the other way round. They are followed by UNPOP, UAUM, SVD, Random and POP in this particular order.
(3) We also see the benefits of transfer learning since *SerRec* outperforms $SerRec_{NoTL}$ for most of the metrics.
(4) We observe that popularity of the items is working against the serendipity because UNPOP shows decent performance whereas POP is the worst (even worse than Random).
(5) As expected, serendipity oriented algorithms SPR and UAUM outperform the relevance oriented SVD.

5 DISCUSSION

Our results mostly corresponded to our expectations and the literature on serendipity in recommender systems, i.e.: a) transfer learning improves serendipity (observation 3), b) serendipity oriented recommendation algorithms outperform relevance oriented ones (observation 5) [9, 12] and c) popularity baseline has the lowest serendipity [8]. Our unexpected finding was that the non-personalized algorithm UNPOP outperforms some personalized algorithms (observation 2), which emphasizes the importance of popularity factor for suggesting serendipitous items. This might suggest that popularity is the most important factor for suggesting serendipitous items and that the traditional artificial serendipity metrics [11] reflect the real world scenario. However, answering these questions is beyond research conducted in this paper.

The limitations are mostly caused by the lack of publicly available datasets containing the necessary data. The dataset contains a relatively small number of serendipitous scores. To increase the number of these scores, we marked some items as non-serendipitous for some users. Although, as we explained in Section 3.3, the chance of the mistake is rather small, some items could have been serendipitous to users, while being marked as non-serendipitous ones.

The performance of our approach can be improved with a different configuration of parameters. We used arbitrary learning rate for training GMF and MLP layers, just considering that the loss function should converge (see Section 3.2). We trained them for a limited number of epochs, used the default number of four negative samples per positive sample and also used the default number of four layers in MLP. It is highly probable that further tuning of such parameters would result in further performance gains for *SerRec*.

6 CONCLUSION

This paper presents *SerRec*, a novel approach to use deep neural networks and transfer learning to generate serendipitous recommendations. We employed the Neural Collaborative filtering [3] framework, that we train using a large dataset with relevance scores and then tune using a smaller serendipity oriented dataset. Our approach shows the benefit of transfer learning and improvements over the state-of-the-art serendipity oriented baselines.

In future work, we would like to explore further tuning of the hyper-parameters (number of layers, epochs, etc) of the utilized deep neural networks, to achieve additional performance gains.

ACKNOWLEDGMENTS

The research was supported by KAUTE Foundation and the European Office of Aerospace Research and Development (Grant No FA9550-17-1-0030)

REFERENCES

[1] Oren Barkan and Noam Koenigstein. 2016. Item2Vec: Neural Item Embedding for Collaborative Filtering.. In *MLSP*. 1–6.
[2] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. 2015. E-commerce in Your Inbox: Product Recommendations at Scale. In *SIGKDD*. 1809–1818.
[3] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. 173–182.
[4] Kalervo Järvelin and Jaana Kekäläinen. 2000. IR Evaluation Methods for Retrieving Highly Relevant Documents. In *SIGIR*. ACM, New York, NY, USA, 41–48.
[5] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[6] Yehuda Koren and Robert Bell. 2015. Advances in collaborative filtering. In *Recommender systems handbook*. Springer, 77–118.
[7] Denis Kotkov, Joseph A. Konstan, Qian Zhao, and Jari Veijalainen. 2018. Investigating Serendipity in Recommender Systems Based on Real User Feedback. In *Proceedings of SAC 2018: Symposium on Applied Computing*. ACM.
[8] Denis Kotkov, Shuaiqiang Wang, and Jari Veijalainen. 2016. A survey of serendipity in recommender systems. *Knowledge-Based Systems* 111 (2016), 180–192.
[9] Qiuxia Lu, Tianqi Chen, Weinan Zhang, Diyi Yang, and Yong Yu. 2012. Serendipitous Personalized Ranking for Top-N Recommendation. In *Proceedings of the The IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology*, Vol. 1. IEEE Computer Society, 258–265.
[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*. 3111–3119.
[11] Tomoko Murakami, Koichiro Mori, and Ryohei Orihara. 2008. Metrics for Evaluating the Serendipity of Recommendation Lists. In *Annual Conference of the Japanese Society for Artificial Intelligence*.
[12] Qianru Zheng, Chi-Kong Chan, and Horace H.S. Ip. 2015. An Unexpectedness-Augmented Utility Model for Making Serendipitous Recommendation. In *Advances in Data Mining: Applications and Theoretical Aspects*. Vol. 9165. Springer International Publishing, 216–230.

PIV

LISTWISE RECOMMENDATION APPROACH WITH NON-NEGATIVE MATRIX FACTORIZATION

by

Gaurav Pandey, Shuaiqiang Wang 2018

Intelligent Decision Technologies 2018

https://doi.org/10.1007/978-3-319-92028-3_3

Listwise Recommendation Approach with Non-negative Matrix Factorization

Gaurav Pandey[1] and Shuaiqiang Wang[2]

[1] University of Jyvaskyla, Finland,
gaurav.g.pandey@jyu.fi,
[2] Data Science Lab, jd.com, China

Abstract. Matrix factorization (MF) is one of the most effective categories of recommendation algorithms, which makes predictions based on the user-item *rating matrix*. Nowadays many studies reveal that the ultimate goal of recommendations is to predict correct rankings of these unrated items. However, most of the pioneering efforts on *ranking-oriented* MF predict users' item ranking based on the original rating matrix, which fails to explicitly present users' preference ranking on items and thus might result in some accuracy loss. In this paper, we formulate a novel listwise user-ranking probability prediction problem for recommendations, that aims to utilize a *user-ranking probability matrix* to predict users' possible rankings on all items. For this, we present *LwRec*, a novel listwise ranking-oriented matrix factorization algorithm. It aims to predict the missing values in the user-ranking probability matrix, aiming that each row of the final predicted matrix should have a probability distribution similar to the original one. Extensive offline experiments on two benchmark datasets against several state-of-the-art baselines demonstrate the effectiveness of our proposal.

Keywords: Recommender systems, Collaborative Filtering, Ranking

1 Introduction

Conventional recommendation algorithms like collaborative filtering follow a rating-oriented paradigm. They generally learn a recommendation model with users' observed historical ratings, using which they predict users' ratings on their unrated items. Nowadays, ranking-oriented recommender systems are receiving increasing attention from both academic communities and industry. Many studies reveal that the ultimate goal of recommendations is to predict correct rankings of these unrated items, and prediction of accurate ranking is more important than predicting accurate rating scores [1, 2]. Accurate prediction of ratings does not necessarily imply improvement in the ranking results.

To elaborate, let us consider items: $\{a, b, c\}$ with their correct ratings R : $\{5, 4, 3\}$ and two rating predictions $P_1 : \{3, 4, 5\}$ and $P_2 : \{3, 2, 1\}$. A rating oriented approach would prefer P_1 over P_2, since predicted ratings in P_1 are more accurate, being closer to the ratings in R. However, the order in P_1 $(a < b < c)$

is completely opposite to the desired order ($a > b > c$). In contrast, a ranking oriented approach would prefer P_2, as it predicts the correct ranking, i.e. ($a > b > c$). This would generate desirable results (correct order of items), in spite of having lower accuracy in predicted ratings.

Given the above argument, some pioneering efforts on *ranking-oriented* recommendation algorithms have been proposed. Due to the effectiveness of matrix factorization (MF) algorithms in rating-oriented recommender systems, a few ranking-oriented MF algorithms have been presented, reporting state-of-the-art results. However, most of the pioneering efforts on *ranking-oriented* MF predict users' item ranking based on rating scores, failing to explicitly present users' preference ranking on items and thus possibly resulting in some accuracy loss.

Therefore, we define a novel listwise user-ranking probability prediction problem for recommendations. We utilize the *listwise user-ranking probability matrix* [3] to explicitly characterize users' preference on items. Given a set of rating scores on items, each ranking on items might be possible, where "correct" rankings (higher scores are ranked at top positions) receive greater probabilities. Thus for each user, the probabilities on users' all possible rankings could formulate as a user-ranking probability matrix, where each element presents a probability that certain user holds certain ranking on items. Thus each row of the user-ranking probability matrix consists of users' probabilities on different item rankings, forming a distribution. With the initial probabilities of users' possible rankings on *their rated items*, the listwise user-ranking probability prediction problem aims to predict probabilities of users' possible rankings on *all items*. Meanwhile, the predictions should satisfy the requirements of probability distributions: each element in the probability matrix should be between 0 and 1, and sum of each row should be 1.

Given a collection of items, there might be a very large number of possible rankings (i.e. $n!$ rankings for n items), resulting in a extremely huge ranking probability matrix and calculations in training. In this study, we only consider the top-k ranked items in rankings, and the size of the matrix could be shrinked significantly, especially the size of the ranking probability matrix is equal to that of the user-item rating matrix when $k = 1$. Based on this matrix, we then present *LwRec*, a novel listwise ranking-oriented MF algorithm, which minimizes the difference between the initial distribution on the known rankings and the final distribution on all items with predictions for each user. Considering the non-negative property for each element, we adapt non-negative MF to implement *LwRec*. Our experimental results on benchmark datasets demonstrate significant performance gains over state-of-art recommender algorithms.

To summarize, our contributions are as follows. (1) We define a novel listwise user-ranking probability prediction problem for recommendations. (2) We present an effective algorithm to solve the problem based on non-negative MF. (3) We achieve significant performance gains against state-of-the-art recommendation algorithms on benchmark datasets.

The rest of the paper is organized as follows. Section 2 briefly presents the related work and Section 3 describes the problem formulation. Then, we explain

the *LwRec* approach in Section 4 followed by experimental setup in Section 5. Finally, Section 6 presents the results and Section 7 concludes the paper.

2 Related Work

This section presents related work for collaborative filtering (CF) recommendation algorithms, which use only the ratings given by the users for the items, and do not need the domain knowledge. They are mainly of two types: rating oriented and ranking oriented. While rating oriented algorithms predict unknown item ratings for each user, ranking oriented algorithms predict item rankings. Both of them can be further categorized as memory-based or model based.

Rating Oriented Algorithms: Memory-based rating oriented algorithms are either user-based CF [4], that utilize similarities between users on the basis of available ratings; or item-based CF [5], that utilize similarities between items. Various advanced versions of this approach have been introduced. For example, SLIM [6] directly learns from the data, a sparse matrix of aggregation coefficients that are analogous to the traditional item-item similarities. FISM [7] learns the item-item similarity matrix as a product of two low-dimensional latent factor matrices. Model-based rating oriented algorithms aim to predict ratings by learning a model from observed ratings. Traditional model of this type is matrix factorization (MF) [8], that uses dimensionality reduction to decrease the distance between predicted and observed rating matrices. Some of the models that are based on matrix factorization are: Probabilistic MF [9], Non-negative MF [10], Factorization Machines [11], Hierarchical Poisson MF [12] and LLORMA [13].

Ranking Oriented Algorithms: EigenRank [14] is a well known ranking oriented memory based CF algorithm that follows the pairwise approach. It employs a greedy aggregation method to aggregate predicted pairwise preferences of items into total ranking. VSRank [15] represents users' pairwise preferences for items by using vector space model and utilizes the relative importances of each pairwise preference. Moreover, various model-based ranking oriented CF algorithms have been introduced that try to optimize a ranking oriented objective function. Some of the notable algorithms of this type are: CLiMF [16], CoFiRank [17], ListCF [3] and GBPR [18].

3 Problem Formulation

3.1 User-Ranking Probability Matrix

Considering m users and n items, for each user there are obviously $n!$ possible rankings of items. Given a set of rating scores on items, each ranking on items might be possible, where "correct" rankings (higher scores are ranked at top positions) receive greater probabilities. The probability of item rankings could be derived with the Plackett-Luce model [19], which is a widely used permutation

(each permutation is actually a ranking) probability model in various domains. Each ranking ρ can be represented as an ordered list $(\rho_1, \rho_2 \dots \rho_n)$, where ρ_i represents the item at the ith position, and positions of the items are unique. Hence, the probability of the ranking ρ can be calculated as:

$$Prob(\rho) = \prod_{i=1}^{n} \frac{\gamma(r_{\rho_i})}{\sum_{j=i}^{n} \gamma(r_{\rho_j})}, \tag{1}$$

where r_{ρ_i} is the rating for the item ρ_i and $\gamma(r) = e^r$.

Since, there are $n!$ rankings of items, which is a large number of rankings even for a small value of n, it makes the computation impractical. Hence, we employ the same approach as Huang et al. [3], that uses an alternative efficient method introduced by Cao et al. [20]. The approach focuses only on top k items in the rankings, leading to $\frac{n!}{(n-k)!}$ different top k sets. So, the probability of the rankings ρ_S whose top-k items are exact $S = \{i_1, i_2 \dots i_k\}$ can be calculated as:

$$Prob(\rho_S) = \prod_{j=1}^{k} \frac{\gamma(r_{i_j})}{\sum_{l=j}^{n} \gamma(r_{i_l})} \tag{2}$$

We have m users and for each user we have $p = \frac{n!}{(n-k)!}$ ranking sets for top k items. Now, we construct the user-ranking probability matrix $\Theta_{m \times p}$. In Θ, each row corresponds to a particular user and contains the probabilities for the p rankings. To clarify, if $Prob_{u_i}(S_j)$ represents the probability of ranking S_j calculated for the user u_i (where $1 \leq j \leq p$ and $1 \leq i \leq m$), then:

$$\begin{aligned} \Theta_{i,j} = & Prob_{u_i}(S_j), \text{if ratings of all items in } S_j \text{ are known,} \\ & \perp, \text{otherwise} \end{aligned} \tag{3}$$

Especially when $k = 1$, i.e. when we consider only the top-1 items in all rankings, the size of Θ is equal to that of the user-item rating matrix. This is because, in this case, $p = \frac{n!}{(n-1)!} = n$.

3.2 Objective and Constraints

Given the matrix of known top k probabilities of items: $\Theta_{m \times p}$, where $p = \frac{n!}{(n-k)!}$, we aim to predict the unknown probabilities, that in turn can be used to generate recommendations. This can be achieved by using a listwise loss function and optimizing it using matrix factorization. For this, we define the following objective and the two related constraints:

Objective: Using two matrices $U_{z \times m}$ and $G_{z \times p}$ that construct the predicted probability matrix $U^\top G$, utilize a listwise loss function and matrix factorization to minimize the distance between Θ and $U^\top G$.

C1: Values in $U^\top G$ should be in the range 0 to 1 (as they are probabilities). i.e. $0 \leq U_{ij} \leq 1, \forall i = 1 \dots z$ and $\forall j = 1 \dots m$.

C2: Sum of each row of $U^\top G$ should be 1 (as a row contains probabilities of rankings for a particular user, that should sum up to 1). i.e. $\sum_{j=1}^{p} (U^\top G)_{ij} = 1, \forall i = 1 \dots m$.

Definition 1 (Listwise User-Ranking Probability Prediction). *Given a user-ranking probability matrix Θ, where each observed element of $\Theta_{i,p}$ indicates certain user's probability for her certain (top-k) preference ranking on her rated items, and each row of Θ forms a probability distribution. The listwise user-ranking probability prediction problem aims to predict each user's probability of her top-k preference ranking on all items, where each row of $U^\top G$ forms a probability distribution as well after prediction, and each user's two distributions, observed and predicted, should be as similar as possible. Formally,*

$$\begin{aligned} &\arg\min_{U,G} \sum_{i=1}^{m} \mathit{diff}\,(\Theta_i, (U^\top G)_i), \\ &\quad \text{s. t. } 0 \le (U^\top G)_{ij} \le 1, i = 1, 2, \ldots, m, \text{ and } j = 1, 2, \ldots, p \text{ (C1)} \\ &\qquad \sum_{j=1}^{p} (U^\top G)_{ij} = 1, i = 1, 2, \ldots, m \text{ (C2)} \end{aligned} \tag{4}$$

Here $\mathit{diff}\,(\Theta_i, (U^\top G)_i)$ is the difference between two distributions Θ_i and $(U^\top G)_i$, i.e. the ith row of the user-ranking probability matrix before and after prediction.

4 Prediction Method

In this section, we present *LwRec* to solve our listwise user-ranking probability prediction problem. We use Kullback-Leibler divergence [21], a commonly used measure for calculating difference between probability distributions, to compute $\mathit{diff}\,(\Theta_i, (U^\top G)_i)$. In *LwRec*, we utilize non-negative matrix factorization (MF) [10] to implement our proposed algorithm, which can generate non-negative elements for U and G. Thus the elements of the user-ranking probability matrix $U^\top G$ are all non-negative. In order to satisfy the constraint C2, we introduce a collection of Lagrange penalty terms in the objective function $\sum_{j=1}^{p} (U^\top G)_{ij} = 1$ where $i = 1 \ldots m$. In standard Lagrange methods, the coefficients of Lagrange penalty terms could be either positive or negative, but in non-negative MF, all of the parameters have to be non-negative. Thus we introduce two non-negative vectors α and β, and regard $(\alpha_i - \beta_i)$ that can be either positive or negative, as the coefficient of the ith Lagrange penalty term to formulate our loss function. Moreover, addressing constraint C2 together with ensuring that the values in $U^\top G$ are non-negative, also satisfies constraint C1 (i.e. values in $U^\top G$ should be in range 0 to 1). The formulation of our loss function can be presented formally as follows:

$$\begin{aligned} L(U, G, \alpha, \beta) = &- \sum_{i=1}^{m} \sum_{j=1, \Theta_{ij} \neq \perp}^{p} \Theta_{ij} \log \frac{(U^\top G)_{ij}}{\sum_{l=1, \Theta_{il} \neq \perp}^{p} (U^\top G)_{il}} \\ &+ \sum_{i=1}^{m} (\alpha_i - \beta_i) \left(\sum_{j=1}^{p} (U^\top G)_{ij} - 1 \right) + \frac{\lambda_1}{2} ||U||^2 + \frac{\lambda_2}{2} ||G||^2, \end{aligned} \tag{5}$$

In Equation 5, the first term represents the main optimization objective from Equation 4, which defines the divergence between $U^\top G$ and observed probability matrix Θ. The second term is the weighted cumulative Lagrange penalty term

for constraint C2. The last two terms are l2-norms of U and G to avoid overfitting, where λ_1 and λ_2 are the respective coefficients. By expanding the *log* in $L(U, G, \alpha, \beta)$ and considering that the sum of each row in Θ is 1, the function can be reformulated as:

$$\begin{aligned} L(U,G,\alpha,\beta) = &\sum_{i=1}^{m} \log\Bigg(\sum_{l=1,\Theta_{il}\neq\perp}^{p} (U^\top G)_{il}\Bigg) - \sum_{i=1}^{m}\sum_{j=1,\Theta_{ij}\neq\perp}^{p} \Theta_{ij}\log\left((U^\top G)_{ij}\right) \\ &+ \sum_{i=1}^{m}(\alpha_i - \beta_i)\Bigg(\sum_{j=1}^{p}(U^\top G)_{ij} - 1\Bigg) + \frac{\lambda_1}{2}||U||^2 + \frac{\lambda_2}{2}||G||^2 \end{aligned} \tag{6}$$

To minimize the loss function using gradient descent, we compute its gradients with respect to the variables U, G, α and β and derive the following updates:

$$\begin{aligned} U_{ia} &\leftarrow U_{ia} - \eta_u\Bigg(\frac{\sum_{l=1,\Theta_{il}\neq\perp}^{p} G_{al}}{\sum_{l=1,\Theta_{il}\neq\perp}^{p}(U^\top G)_{il}} - \sum_{j=1,\Theta_{ij}\neq\perp}^{p}\frac{\Theta_{ij}G_{aj}}{(U^\top G)_{ij}} \\ &\quad + (\alpha_i - \beta_i)\sum_{j=1}^{p} G_{aj} + \lambda_1 U_{ia}\Bigg), \\ G_{aj} &\leftarrow G_{aj} - \eta_g\Bigg(\sum_{i=1}^{m}\frac{U_{ia}}{\sum_{l=1,\Theta_{il}\neq\perp}^{p}(U^\top G)_{il}} - \sum_{i=1,\Theta_{ij}\neq\perp}^{m}\frac{\Theta_{ij}U_{ia}}{(U^\top G)_{ij}} \\ &\quad + \sum_{i=1}^{m}(\alpha_i - \beta_i)U_{ia} + \lambda_2 G_{aj}\Bigg), \\ \alpha_i &\leftarrow \alpha_i - \eta_\alpha\Bigg(\sum_{j=1}^{p}(U^\top G)_{ij} - 1\Bigg) \text{ and } \beta_i \leftarrow \beta_i - \eta_\beta\Bigg(1 - \sum_{j=1}^{p}(U^\top G)_{ij}\Bigg) \end{aligned} \tag{7}$$

where, η_u, η_g, η_α and η_β are the step sizes. Now, using non-negative matrix factorization [10], we choose the step sizes such that:

$$\begin{aligned} \eta_u &= \frac{U_{ia}}{\frac{\sum_{l=1,\Theta_{il}\neq\perp}^{p} G_{al}}{\sum_{l=1,\Theta_{il}\neq\perp}^{p}(U^\top G)_{il}} + \alpha_i\sum_{j=1}^{p} G_{aj} + \lambda_1 U_{ia}}, \\ \eta_g &= \frac{G_{aj}}{\sum_{i=1}^{m}\frac{U_{ia}}{\sum_{l=1,\Theta_{il}\neq\perp}^{p}(U^\top G)_{il}} + \sum_{i=1}^{m}\alpha_i U_{ia} + \lambda_2 G_{aj}}, \\ \eta_\alpha &= \frac{\alpha_i}{\sum_{j=1}^{p}(U^\top G)_{ij}} \text{ and } \eta_\beta = \beta_i \end{aligned} \tag{8}$$

Substituting these values of the steps in updation formulas in Equations 7, we derive the following multiplicative updates:

$$\begin{aligned} U_{ia} &\leftarrow U_{ia}\frac{\sum_{j=1,\Theta_{ij}\neq\perp}^{p}\frac{\Theta_{ij}G_{aj}}{(U^\top G)_{ij}} + \beta_i\sum_{j=1}^{p} G_{aj}}{\frac{\sum_{l=1,\Theta_{il}\neq\perp}^{p} G_{al}}{\sum_{l=1,\Theta_{il}\neq\perp}^{p}(U^\top G)_{il}} + \alpha_i\sum_{j=1}^{p} G_{aj} + \lambda_1 U_{ia}}, \\ G_{aj} &\leftarrow G_{aj}\frac{\sum_{i=1,\Theta_{ij}\neq\perp}^{m}\frac{\Theta_{ij}U_{ia}}{(U^\top G)_{ij}} + \sum_{i=1}^{m}\beta_i U_{ia}}{\sum_{i=1}^{m}\frac{U_{ia}}{\sum_{l=1,\Theta_{il}\neq\perp}^{p}(U^\top G)_{il}} + \sum_{i=1}^{m}\alpha_i U_{ia} + +\lambda_2 G_{aj}}, \\ \alpha_i &\leftarrow \frac{\alpha_i}{\sum_{j=1}^{p}(U^\top G)_{ij}} \text{ and } \beta_i \leftarrow \beta_i\sum_{j=1}^{p}(U^\top G)_{ij} \end{aligned} \tag{9}$$

Algorithm 1: *LwRec* Algorithm

Input: Ratings for n items by m users, values k and z

Initialize $\Theta_{m \times p}$, where $p = \frac{n!}{(n-k)!}$ (See Equation 3)

Randomly initialize $U_{z \times m}, G_{z \times p}$, α_m and β_m with non-negative values

repeat

  Update U, G, α and β according to Equation 9

until *Reach convergence or the max iteration*;

return $U^\top G$

On optimizing U and G, The rows of $U^\top G$ would contain predicted probability distributions of top k rankings for users, that can be utilized to generate recommendations. Algorithm 1 summarizes our method.

5 Experimental Setup

5.1 Datasets

For our experiments, we use two MovieLens[3] data sets: MovieLens-100K and MovieLens-1M. MovieLens-100K dataset contains 100,000 ratings given by 943 users on 1682 movies. MovieLens-1M dataset is larger with 1,000,000 ratings given by 6040 users on 3952 movies. In MovieLens-100K as well as MovieLens-1M the ratings are given on an integer scale from 1 to 5. For both the datasets we assign 10 ratings for each user for testing and the rest for training.

5.2 *LwRec* Setup

For both datasets, we consider top 1 item rankings (i.e. $k = 1$), since the topmost position in a ranking is the most important one. Moreover, it also makes our experiments computationally inexpensive since when $k = 1$, $p = \frac{n!}{(n-1)!} = n$ (number of items). A higher value of k, would make the value of p huge. For example, for MovieLens-1M ($n = 3952$), when $k = 2$, $p \approx 1.56 \times 10^7$ and for $k = 3$, $p \approx 6.17 \times 10^{10}$. Probably higher values of k could result in some performance gains, but in this study we restrict the scope to experiment with $k = 1$. Moreover, for the matrices U and G, we have used the column length of 10 (i.e. $z = 10$).

We generate the probability distributions of known item rankings i.e. Θ using the training set and then generate the matrix of predicted probabilities i.e. $U^\top G$ (Algorithm 1). Since we use $k = 1$, each row of $U^\top G$ would contain probabilities for n items (as each item ranking has only one item in this case). Therefore, items in the test set can be simply ordered by their decreasing predicted probabilities.

5.3 Baselines

We used the following state-of-the-art algorithms as our comparison partners:

[3] http://grouplens.org/datasets/movielens/

1. CF: CF [22] calculates the similarity between users, and ranks the items according to the predicted ratings for each user.
2. Matrix Factorization (MF): User matrix U and item matrix I are optimized in MF [8], to minimize the difference between their product UI^T and rating matrix R. UI^T regenerates the rating matrix to predict unknown ratings.
3. EigenRank: EigenRank [14] is a pair-wise ranking-oriented algorithm that employs a greedy aggregation method to aggregate the predicted pairwise preferences of items into total ranking.
4. ListRankMF: ListRankMF [23] minimizes a loss function representing uncertainty between training and output lists produced by a MF ranking model.
5. FISM: Factored Item Similarity Models (FISM) [7] learn the item-item similarity matrix as a product of two low-dimensional latent factor matrices. While **FISMrmse** computes loss using sqaured error loss function, **FISMauc** considers a ranking error based loss function.
6. LLORMA: Local Low-Rank Matrix Approximation (LLORMA) [13] approximates the observed matrix as a weighted sum.
7. ListCF: ListCF [3], a ranking oriented CF algorithm, predicts item order for a user, based on similar users probability distributions over item permutations.

5.4 Evaluation Metrics.

We use the standard ranking accuracy metric called normalized discounted cumulative gain (NDCG@1-10) [24] that is able to handle multiple levels of relevance, to evaluate item rankings generated by *LwRec* and the baselines.

Statistical significance of observed differences between the performance of two runs is tested using a two-tailed paired t-test and is denoted using $^{\blacktriangle}$ (or $^{\blacktriangledown}$) for strong significance for $\alpha = 0.01$; or $^{\triangle}$ (or $^{\triangledown}$) for weak significance for $\alpha = 0.05$.

6 Results

In Table 1, we can see that *LwRec* outperforms the comparison partners for all the metrics (NDCG@1 to 10) for MovieLens-100K as well as MovieLens-1M. ListCF is the second best followed by LLORMA and FISMrmse, for both datasets. For MovieLens-100K, EigenRank and ListRankMF have comparable performances followed by MF and FISMauc. For MovieLens-1M, ListRankMF performs better than FISMauc followed by MF.

We also calculate statistical significance of *LwRec* against ListCF which is our best performing comparison algorithm. The results for MovieLens-100K show weak to strong statistical significance for most metrics and for MovieLens-1M the results have strong statistical significance in almost all cases.

7 Conclusion

In this paper, we defined a novel listwise user-ranking probability prediction problem. Then we described *LwRec*, a listwise recommendation algorithm, that

Table 1. Ranking Performance of *LwRec* against baselines

Statistical significance shown for *LwRec* against LLORMA

Performance for MovieLens-100K										
NDCG	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10
CF	0.5990	0.6394	0.6707	0.6938	0.7182	0.7442	0.7705	0.7970	0.8245	0.8546
MF	0.6629	0.6711	0.6918	0.7158	0.7373	0.7651	0.7895	0.8154	0.8418	0.8683
EigenRank	0.6734	0.6799	0.6972	0.7192	0.7408	0.7634	0.7889	0.8146	0.8407	0.8701
ListRankMF	0.6769	0.6792	0.6989	0.7140	0.7316	0.7532	0.7772	0.8057	0.8368	0.8684
FISMauc	0.6480	0.6681	0.6912	0.7132	0.7363	0.7598	0.7826	0.8086	0.8360	0.8661
FISMrmse	0.6735	0.6868	0.7060	0.7246	0.7475	0.7684	0.7914	0.8164	0.8431	0.8726
LLORMA	0.6794	0.6898	0.7092	0.7264	0.7488	0.7705	0.7950	0.8219	0.8462	0.8738
ListCF	0.6846	0.6897	0.7100	0.7274	0.7500	0.7732	0.7982	0.8243	0.8499	0.8752
LwRec	**0.6930**	**0.6991**	**0.7200**$^{\triangle}$	**0.7422**$^{\blacktriangle}$	**0.7643**$^{\blacktriangle}$	**0.7844**$^{\blacktriangle}$	**0.8059**$^{\blacktriangle}$	**0.8287**	**0.8527**	**0.8801**$^{\triangle}$
Performance for MovieLens-1M										
NDCG	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10
CF	0.6214	0.6498	0.6710	0.6954	0.7189	0.7437	0.7708	0.7981	0.8272	0.8589
MF	0.6619	0.6649	0.6802	0.7008	0.7238	0.7483	0.7741	0.8026	0.8322	0.8642
EigenRank	0.6486	0.6571	0.6746	0.6958	0.7190	0.7428	0.7688	0.7966	0.8268	0.8608
ListRankMF	0.7084	0.7078	0.7203	0.7342	0.7532	0.7736	0.7972	0.8225	0.8498	0.8803
FISMauc	0.6784	0.6951	0.7109	0.7315	0.7526	0.7750	0.7983	0.8235	0.8493	0.8770
FISMrmse	0.7157	0.7178	0.7279	0.7440	0.7634	0.7849	0.8071	0.8315	0.8569	0.8847
LLORMA	0.7116	0.7174	0.7303	0.7479	0.7672	0.7878	0.8100	0.8340	0.8587	0.8854
ListCF	0.7204	0.7243	0.7359	0.7504	0.7685	0.7895	0.8136	0.8384	0.8627	0.8876
LwRec	**0.7204**	**0.7281**	**0.7428**$^{\blacktriangle}$	**0.7600**$^{\blacktriangle}$	**0.7777**$^{\blacktriangle}$	**0.7988**$^{\blacktriangle}$	**0.8207**$^{\blacktriangle}$	**0.8436**$^{\blacktriangle}$	**0.8667**$^{\blacktriangle}$	**0.8906**$^{\blacktriangle}$

solves the problem by minimizing a listwise loss function using non-negative matrix factorization. Our experimental results on benchmark datasets show significant performance gains of *LwRec* over state-of-the-art recommender algorithms.

In this study, we have experimented for top k item rankings, for $k = 1$. In the future, we would like to explore the effect on results on using higher value of k. Moreover, we have used column length 10 for the matrices U and G. It would be interesting to see the changes in results on varying this column length.

References

1. Gunawardana, A., Shani, G.: A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. J. Mach. Learn. Res. **10** (2009) 2935–2962
2. Adomavicius, G., Tuzhilin, A.: Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Trans. on Knowl. and Data Eng. **17**(6) (2005) 734–749
3. Huang, S., Wang, S., Liu, T.Y., Ma, J., Chen, Z., Veijalainen, J.: Listwise Collaborative Filtering. In: SIGIR. (2015) 343–352
4. Herlocker, J., Konstan, J.A., Riedl, J.: An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms. Inf. Retr. **5**(4) (2002) 287–310
5. Linden, G., Smith, B., York, J.: Amazon.com Recommendations: Item-to-item Collaborative Filtering. IEEE Internet Comput. **7**(1) (2003) 76–80

6. Ning, X., Karypis, G.: SLIM: Sparse Linear Methods for Top-N Recommender Systems. In: ICDM. (2011) 497–506
7. Kabbur, S., Ning, X., Karypis, G.: FISM: Factored Item Similarity Models for top-N Recommender Systems. In: SIGKDD. (2013) 659–667
8. Koren, Y., Bell, R., Volinsky, C.: Matrix Factorization Techniques for Recommender Systems. Computer **42**(8) (2009) 30–37
9. Salakhutdinov, R., Mnih, A.: Probabilistic Matrix Factorization. In: NIPS. (2007) 1257–1264
10. Lee, D.D., Seung, H.S.: Algorithms for Non-negative Matrix Factorization. In: NIPS. (2000) 556–562
11. Rendle, S.: Factorization Machines with libFM. ACM Trans. Intell. Syst. Technol. **3**(3) (2012) 57:1–57:22
12. Gopalan, P., Hofman, J.M., Blei, D.M.: Scalable Recommendation with Hierarchical Poisson Factorization. In: UAI. (2015) 326–335
13. Lee, J., Kim, S., Lebanon, G., Singer, Y.: Local Low-rank Matrix Approximation. In: ICML. (2013) II–82–II–90
14. Liu, N.N., Yang, Q.: EigenRank: A Ranking-oriented Approach to Collaborative Filtering. In: SIGIR. (2008) 83–90
15. Wang, S., Sun, J., Gao, B.J., Ma, J.: VSRank: A Novel Framework for Ranking-Based Collaborative Filtering. ACM Trans. Intell. Syst. Technol. **5**(3) (2014) 51:1–51:24
16. Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Oliver, N., Hanjalic, A.: CLiMF: Learning to Maximize Reciprocal Rank with Collaborative Less-is-more Filtering. In: RecSys. (2012) 139–146
17. Weimer, M., Karatzoglou, A., Le, Q.V., Smola, A.: CofiRank: Maximum margin matrix factorization for collaborative ranking. In: NIPS. (2007) 1593–1600
18. Pan, W., Chen, L.: GBPR: Group Preference Based Bayesian Personalized Ranking for One-class Collaborative Filtering. In: IJCAI. (2013) 2691–2697
19. Marden, J.I.: Analyzing and Modeling Rank Data. CRC Press (1996)
20. Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H.: Learning to Rank: From Pairwise Approach to Listwise Approach. In: ICML. (2007) 129–136
21. Kullback, S.: Information Theory And Statistics. Dover Pubns (1997)
22. Breese, J.S., Heckerman, D., Kadie, C.: Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In: UAI. (1998) 43–52
23. Shi, Y., Larson, M., Hanjalic, A.: List-wise Learning to Rank with Matrix Factorization for Collaborative Filtering. In: RecSys. (2010) 269–272
24. Järvelin, K., Kekäläinen, J.: Cumulated Gain-based Evaluation of IR Techniques. ACM Trans. Inf. Syst. **20**(4) (2002) 422–446

PV

CITYSEARCHER: A CITY SEARCH ENGINE FOR INTERESTS

by

Mohamed Abdel Maksoud, Gaurav Pandey (first co-author), Shuaiqiang Wang
2017

Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval

CitySearcher: A City Search Engine For Interests

Mohamed Abdel Maksoud*
Codoma.tech Advanced Technologies
Egypt
mohamed@amaksoud.com

Gaurav Pandey*
University of Jyvaskyla
Finland
gaurav.g.pandey@student.jyu.fi

Shuaiqiang Wang
University of Manchester
UK
shuaiqiang.wang@manchester.ac.uk

ABSTRACT

We introduce *CitySearcher*, a vertical search engine that searches for cities when queried for an interest. Generally in search engines, utilization of semantics between words is favorable for performance improvement. Even though ambiguous query words have multiple semantic meanings, search engines can return diversified results to satisfy different users' information needs. But for *CitySearcher*, mismatched semantic relationships can lead to extremely unsatisfactory results. For example, the city *Sale* would incorrectly rank high for the interest *shopping* because of semantic interpretations of the words. Thus in our system, the main challenge is to eliminate the mismatched semantic relationships resulting from the side effect of the semantic models. In the previous case, we aim to ignore the semantics of a city's name which is not indicative of the city's characteristics. In *CitySearcher*, we use *word2vec*, a very popular word embedding technique to estimate the semantics of the words and create the initial ranks of the cities. To reduce the effect of the mismatched semantic relationships, we generate a set of features for learning based on a novel clustering-based method. With the generated features, we then utilize learning to rank algorithms to rerank the cities for return. We use the English version of Wikivoyage dataset for evaluation of our system, where we sample a very small dataset for training. Experimental results demonstrate the performance gain of our system over various standard retrieval techniques.

CCS CONCEPTS

• **Information systems → Retrieval tasks and goals**;

KEYWORDS

Information Retrieval, word2vec, Feature Engineering

ACM Reference format:
Mohamed Abdel Maksoud, Gaurav Pandey, and Shuaiqiang Wang. 2017. CitySearcher: A City Search Engine For Interests. In *Proceedings of SIGIR '17, Shinjuku, Tokyo, Japan, August 07-11, 2017*, 4 pages.
https://doi.org/10.1145/3077136.3080742

*Indicates equal contribution

SIGIR '17, August 07-11, 2017, Shinjuku, Tokyo, Japan

ACM ISBN 978-1-4503-5022-8/17/08…$15.00
https://doi.org/10.1145/3077136.3080742

1 INTRODUCTION

In this paper, we present *CitySearcher*, a vertical search engine that generates a ranking of cities in response to an interest given by the user. The interest acts as a query on a travel document corpus, which mostly contains documents that represent and contain information about a particular city. For example, for the interest *'History'* the top 3 cities presented are *'Rome'*, *'Lviv'* and *'Oslo'*.

Generally in search engines, utilization of semantics between words is beneficial for an information retrieval system for performance improvement. To illustrate, the presence of sentences like *'The city has bright sea shores'* in a document should contribute in giving it a high ranking score for the query *'sunny beach'*. Even though the sentence and the query have no common words, but there is semantic similarity between the words *'bright'* and *'sunny'* as well as *'shores'* and *'beach'*. Even though ambiguous query words have multiple semantic meanings, search engines can return diversified results to satisfy different users' information needs.

However in *CitySearcher*, mismatched semantic relationships can lead to extremely unsatisfactory results. This happens particularly when the city names gets incorrectly semantically interpreted. As an example, the city *'Sale'* would incorrectly rank high for an interest like *'shopping'*, because of semantic relationships between the two words as well as repetition of the city's name in its document. Thus for *CitySearcher*, the main challenge is to eliminate the mismatched semantic relationships resulting from the side effect of the semantic models. In the previous case, it is rather desired that the semantics of city name are ignored completely for matching it with the interest. This is because, the meaning of a city name if any, can be completely unrelated to its characteristics.

In *CitySearcher*, we we use *word2vec* [7, 8] to estimate the semantics of the words. *Word2vec* is a widely used word embedding technique, and it is shown that the vectors generated by *word2vec* preserve the semantic relations between the words, *e.g. vec('Paris') - vec('France') + vec('Italy')* is very close to *vec('Rome')* [7].

With the semantic vectorizations of the words, the *CitySearcher* algorithm firstly represents each city as the words in the corresponding document describing the target city, and then calculates the initial ranking scores for each city-interest pair with the similarities between the vectors of the words in the city document and the interest. While *word2vec* provides semantic interpretations in retrieval for *CitySearcher*, it also introduces the issue of certain mismatched semantic similarities, especially between interest and city names. Thus besides the initial ranking scores between cities and interest queries, we also propose a set of new features for ranking of the cities with the given interest queries. In particular, we create a set of topics by clustering all of the vectors of words in the vocabulary. For each interest, we choose a subset of closest topics and calculate the similarity from the city to the topics as new

features. We finally create a training set by collecting users' relevance assessments (ratings) for city-interest pairs, and use machine learning (e.g. different regression algorithms) to rerank the results of the cities for return.

While *CitySearcher* could preserve semantic relationships in general and improve retrieval performance, it is also expected to show capability of reducing the effect of undesired semantic relationships. With our proposed features, the semantics of the interest query words are interpreted with a set of closest topics (clusters of words), allowing us calculating the similarity between cities and interest queries with multiple enriched dimensions.

We use the English version of Wikivoyage dataset [1] for evaluation of our system. With a very small sampled dataset for training, experimental results demonstrate the performance gain of our system over various standard retrieval techniques.

2 PROBLEM STATEMENT

Let D be a corpus containing travel documents doc_{c_1}, doc_{c_2} ... doc_{c_z} having information about the cities $c_1, c_2 \dots c_z$, respectively. There is one-to-one mapping between documents and cities, i.e. each document represents only one city and each city has only one document for it. For a word representing an interest itr, retrieve a ranking of cities in decreasing order of relevance.

The main challenges or targets for the retrieval task along with their proposed solutions are:

(a) The semantic relationships between words are beneficial for retrieval, and hence should be utilized.
Solution: Create vector representations of the words in corpus using *word2vec* (that preserves semantic relationships) and then use them for retrieval.

(b) However, consideration of semantic similarity between certain words, especially city and interest, lead to irrelevant results. Such effect mismatched of semantic relationships needs to be reduced.
Solution: To reduce the effect of the undesired semantic relationships, train a ranking model using machine learning. For the training process, collection of ratings from the users for city-document pairs is a prerequisite.

(c) There are few features for city-document pairs as well as limited number of ratings from the users. This limits the efficiency of training.
Solution: Generate new features for city-document pairs using vector representations of words in the vocabulary.

3 METHOD

Mokolov et al [7, 8] introduced *word2vec*, a prominent procedure to generate vector representations of words. They introduced the continuous *skip-gram*, which is a neural network model consisting of input, projection and output layers that predict the surrounding words. This is achieved by maximizing the average log probability for the words $(w_1, w_2 \dots w_N)$ present in the corpus:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{j \in Sur(i,z)} log\, p(w_j|w_i) \tag{1}$$

[1] https://en.wikivoyage.org/wiki/Main_Page

Here, $Sur(i,z)$ represents a context window for training consisting of z words that surround the word i. Also, $p(w_j|w_i)$ is the hierarchical softmax of the respective word vectors $v(w_j)$ and $v(w_i)$. One of the key benefits of generating word vectors is that the training is completely unsupervised. Moreover, the generated vectors enable the calculation of similarities between words, by simply calculating the similarity of vectors.

CitySearcher utilizes *word2vec* on a corpus comprised mainly of travel destination documents (each document has information about a particular city), so that vector representations of each word present in the vocabulary is computed. That is to say, for all the words present in the corpus: $w_1, w_2 \dots w_N$, the corresponding vector representations: $v(w_1), v(w_2) \dots v(w_N)$ are generated. The sections that follow, describe how these vector representations are used to generate ranking of cities for an interest.

3.1 Basic Algorithm: Using Vector Similarity

The similarity between two words $w1$ and w_2 from the corpus can be calculated on the basis of cosine distance between their vectors:

$$sim(w_1, w_2) = 1 - cosineDistance(v(w_1), v(w_2)) \tag{2}$$

To rank cities against an interest itr (a word representing an interest, e.g. music, history etc.), we need to find the ranking score for itr and the cities. This can be done by calculating the average similarity score for itr and the words closest to it in the city document. If doc_c is the document representing the city c and contains words $w_1, w_2 \dots w_n$, the similarity scores are calculated between itr and the words of the document: $sim(itr, w_1)$, $sim(itr, w_2)$ ... $sim(itr, w_n)$. Let $s_1, s_2 \dots s_k$ be the top k scores from these similarity scores. Then, the ranking score for the city-interest pair is calculated as the average of top-k similarities. This ranking score, $initialScore(c, itr)$ can be formulated as:

$$initialScore(c, itr) = \frac{1}{k}\sum_{i=1}^{k} s_i \tag{3}$$

The calculation of such scores for an interest corresponding to different cities, and then sorting the cities in a deacreasing order of the scores, enable ranking the cities for the interest.

3.2 Improvement using Machine Learning

The algorithm described in the previous section utilizes the semantic relationship between words, captured by their vector representations. Though this is expected to perform well in most cases, utilization of vector similarities can lead to fundamental problems in particular situations. Since the city's name is semantically interpreted and it is usually mentioned several times in the document, it has a significant influence on the ranking. As a result, a city called *Sale* ranks first for the interest *Shopping*. Similarly, a small village named *Chicken* appears in the top documents for the interest *Food* and a city called *Mobile* ranks highly for the interest *Technology*.

To circumvent these defects and improve the performance, the city rankings could be generated by training using regression algorithms like Kernel ridge regression [9] and Logistic regression [9], on the relevant assessments obtained from the users. A relevance assessment can be represented as a tuple: (c (*city*), itr (*interest*), r

(*rating*)) where r $\in \{1, -1\}$. The user gives a value 1 to r if she thinks that city c is relevant for the interest itr, and -1 if it is irrelevant. However, there is only one feature derived from vector representations that can be used in training i.e. the city-interest initial ranking score as calculated in Equation 3. The lack of features would limit the effectiveness of the training. Hence, enriching our training dataset would improve training effectiveness. To this end, in the next section we describe a novel approach employed by us, to generate a set of new features per relevance assessment, by utilizing vector representations of the words in the vocabulary.

3.3 CitySearcher Feature Generation

Our novel technique generates features corresponding to each relevance assessment (c, itr, r). We create a set of topics by clustering all of the vectors of words in the vocabulary. For the interest itr, we choose a subset of closest topics and calculate the similarity from the city c to these topics as new features. The intuition behind using this technique is that the relevance of a c to itr, should be related to the similarity of c to the topics closest to itr.

Firstly we use k-means [12] clustering algorithm to divide the entire vocabulary into M clusters. The clustering algorithm utilizes the precomputed vector representations of the words in the corpus and calculates the similarity between the words using Equation 2. The centroid of word vectors in each cluster is considered as a topic (also a vector), resulting in creating M topics. Also, each topic is representative of the words in its cluster. Thereafter, for the relevance assessment (c, itr, r) we find t_{itr}, the topic to whose cluster the interest word itr belongs. Then, we find the p topics closest to t_{itr}: $t_1, t_2 \ldots t_p$, which are arranged in increasing order of their cosine-distance from t_{itr}. It should be noted that the p closest topics to t_{itr} also includes t_{itr} itself, making $t_1 = t_{itr}$.

Since in our case, each city c is represented by only one document doc_c and also doc_c is representative only for c, the calculation of city to topic similarity becomes quite straightforward. It can be calculated as $topicSimilarity(c, t)$, which is the ratio of the number of words in doc_c that belong to the cluster for topic t:

$$topicSimilarity(c, t) = \frac{|(words\ in\ doc_c) \cap (words\ in\ cluster\ of\ t)|}{|words\ in\ doc_c|} \tag{4}$$

For the relevance assessment (c, itr, r), the similarities of c to the p closest topics to t_{itr} $(t_1, t_2 \ldots t_p)$ can be considered as the p newly generated features $(f_1, f_2 \ldots f_p)$:

$$f_i = topicSimilarity(c, t_i), i = 1, 2, \ldots p \tag{5}$$

It should be noted that other features can also be included for training along with the generated features. In our method we include two more features: (a) initial ranking score for city document pair, $initialScore(c, itr)$, using the basic approach (Section 3.1) (b) document length, as it is a strong indicator of importance for travel destination documents. This makes a total of $p + 2$ features. These features are used to form the training data, that in turn is used by regression algorithms, that generates improved ranking models.

3.4 Remarks

We have used a rather sophisticated method to construct the training set. The method was dictated by the relatively small number of ratings available. Potentially better results can be constructed if we build a dataset for each interest. In such setting, the ratios for each topic would have the same order throughout the dataset, which makes learning a model out of them more straightforward.

4 EXPERIMENTAL SETUP

4.1 Dataset

We have used the English version of Wikivoyage dataset for our experiments, which contains 6691 documents that predominantly represent different travel destinations (cities). It has a corpus size of 1832499 words and a vocabulary of 106634 words. Also, we created a list of 50 common travel interests, by taking inspiration from several famous travel webpages. Moreover, for relevance assessment, we collected 800 ratings for randomly selected city-interest pairs. These ratings were received from 40 people (Europeans, both genders, age 18-60) using the crowdsourcing platform clickworker [2]. Furthermore to enable training and testing, the ratings were divided into a training set (80%) and a test set (20%).

4.2 Evaluation Metric

We use the standard ranking accuracy metric: normalized discounted cumulative gain (NDCG@1-5) [6], to evaluate the rankings generated by our algorithms and the baselines.

4.3 CitySearcher Setup

Vector representations for the words in WikiVoyage dataset are created using *word2vec*, for a window size of 10. Firstly, the rankings of cities are generated for the interests by using the basic *CitySearcher* algorithm described in Section 3.1. For calculating the ranking scores, we used top 10 similarity scores between the interest and the words in the document, i.e. $k = 10$.

Then, to implement the feature generation technique described in Section 3.3 to enrich the training set, we clustered the word vectors and created 100 topics for the Wikivoyage dataset (i.e. $M = 100$). Thereafter, for each relevant assessment (c, itr, r), we generated features corresponding to all the 100 closest topics to the interest itr, i.e. value of p is also 100. It means that the similarities of c are calculated to all the topics (ordered from closest to farthest to the topic of interest itr), and considered as features. Two more features were included: the document length for doc_c (the document representing c) and the initial ranking score for city-interest pair using Equation 3. Thus, for each relevance assessment we have a total of 102 features. Admittedly, these values of k, M and p have been chosen intuitively.

The following two regression algorithms were applied on the training set with generated features to create the ranking models:

- **Kernel Ridge Regression**: Kernel ridge regression (KRR) [9] combines Ridge Regression (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function in the space induced by the respective kernel and the data. We used RBF kernels. The optimal parameters were found using 3-fold cross validation.

[2] https://clickworker.com

Table 1: Performance comparison for NDCG@1-5

NDCG	@1	@2	@3	@4	@5
TF-IDF	0.8000	0.7500	0.8055	0.8017	0.8234
Okapi-BM25	0.8000	0.8500	0.8380	0.8479	0.8542
LSI	0.8000	0.7500	0.8055	0.8017	0.8234
LDA	0.8000	0.7500	0.7620	0.7521	0.7821
LGD	**0.9000**	0.8000	0.8480	0.8403	0.8475
CS-Basic	0.8000	0.7500	0.7740	0.7861	0.8154
CS-KRR	**0.9000**	**0.9000**	0.8161	0.8465	0.8620
CS-LR	0.8000	0.8500	**0.8576**	**0.8614**	**0.8638**

- **Logistic Regression**: Logistic regression [5] is a classification algorithm, that nevertheless computes estimates of the class probabilities, that can be used for ranking.

For these regression algorithms, we have used the implementation provided by the Python package scikit-learn [3]. The models generated by these models are evaluated on the test set.

4.4 Baselines

To compare the rankings produced by our algorithms, we use the five widely used retrieval techniques as baselines:

- **TF-IDF**: Term frequency- Inverse Document Frequency [10] is a very famous ranking model and employs bag-of-words representation. The weight of a term increases proportionally to the number of its occurrences in the document, but also decreases in proportion to its frequency in the corpus.
- **Okapi-BM25**: Okapi BM25 [11], a well known ranking scheme, is based on probabilistic retrieval framework.
- **LSI**: Latent Semantic Indexing [4] uses singular value decomposition to identify patterns in the relationships between the terms to generate a semantic feature space.
- **LDA**: Latent Dirichlet Allocation [1] models each document using the underlying set of word topics.
- **LGD**: LGD [2, 3] weighting model is a high performing version of the log-logistic model.

5 RESULTS

The results are shown in Table 1 for the metric NDCG@1-5. The results are presented for the baselines: TF-IDF, Okapi BM25, LSI, LDA and LGD. Also, the basic *CitySearcher* algorithm from Section 3.1 is denoted as CS-Basic. Moreover, the evaluations for the learning-to-rank models trained on generated features (Section 3.3) using Kernel Ridge Regression and Logistic Regression are presented as CS-KRR and CS-LR respectively.

We can see that the results for CS-Basic are better than the baseline LDA for all metrics, however they are worse than TF-IDF, Okapi-BM25, LSA and LGD in most cases. Also, LGD is the best performing method among the baselines. We observe the benefits of using machine learning on generated features, as the performances of both such methods, CS-KRR and CS-LR, are better than that of CS-Basic. CS-LR gives the best results for NDCG@3-5, while CS-KRR gives the best performance for NDCG@1-2. However, LGD ties with CS-KRR on NDCG@1. Overall, we can conclude that CS-LR is the best performing method because apart from giving best performances for NDCG@3-5, its performance is better than most baselines even for NDCG@1-2. CS-KRR on the other hand hand falls behind LGD as well as Okapi-BM25 for NDCG@3-5.

6 CONCLUSION

We introduced *CitySearcher*, a search engine that ranks cities for interests. It uses vector representations of words to estimate their semantics. The basic algorithm computes ranking scores for each city-interest pair using similarities between the vectors, but suffers because of mismatched semantic similarities. To solve this issue, we propose a set of new features to rerank cities for the interest. A set of topics is created by clustering all vectors of words in the vocabulary. Then, we choose a subset of closest topics to the interest and calculate the similarity from the city to the topics as new features. Even for a small training set, the results show the benefits of reranking using regression algorithms on generated features over the basic algorithm as well as the standard retrieval techniques.

In future work, we want to create an algorithm that can incorporate more than one interest in the query. Additionally, we would like to return a ranking of tours (group of cities), in response to multiple interests. Since we have a small training set in our experiments, we would like to experiment with larger training data. Moreover, the parameters in our experiments can be further explored and tuned to achieve further improvements.

ACKNOWLEDGMENTS

This work was supported by Codoma.tech Advanced Technologies in the context of the Travición project[4], the Academy of Finland (MineSocMed Grant No 268078) and the Natural Science Foundation of China (Grant No 71402083).

REFERENCES

[1] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3 (2003), 993–1022.
[2] Stéphane Clinchant and Eric Gaussier. 2009. Bridging Language Modeling and Divergence from Randomness Models: A Log-Logistic Model for IR. In *Proceedings of the 2Nd International Conference on Theory of Information Retrieval: Advances in Information Retrieval Theory*. Springer-Verlag, 54–65.
[3] Stéphane Clinchant and Eric Gaussier. 2010. Information-based Models for Ad Hoc IR. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 234–241.
[4] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science* 41, 6 (1990), 391–407.
[5] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied Logistic Regression*. Vol. 398. John Wiley & Sons.
[6] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-based Evaluation of IR Techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.
[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013).
[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.
[9] Kevin P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective*. The MIT Press.
[10] Gerard Salton and Christopher Buckley. 1988. Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management* 24, 5 (1988), 513–523.
[11] Amit Singhal. 2001. Modern Information Retrieval: A Brief Overview. *IEEE Data Eng. Bull.* 24, 4 (2001), 35–43.
[12] Douglas Steinley and Michael J. Brusco. 2007. Initializing K-means Batch Clustering: A Critical Evaluation of Several Techniques. *Journal of Classification* 24, 1 (2007), 99–121.

[3] http://scikit-learn.org/

[4] http://www.travicion.com

PVI

FINDING TOURS FOR A SET OF INTERESTS

by

Mohamed Abdel Maksoud, Gaurav Pandey (first co-author), Shuaiqiang Wang
2018

Companion Proceedings of The Web Conference 2018

Finding Tours for a Set of Interests

Mohamed Abdel Maksoud*
Codoma.tech Advanced Technologies
Egypt
mohamed@amaksoud.com

Gaurav Pandey*
University of Jyvaskyla
Finland
gaurav.g.pandey@jyu.fi

Shuaiqiang Wang
Data Science Lab, jd.com
China
wangshuaiqiang1@jd.com

ABSTRACT

This paper addresses a novel tour discovery problem in the domain of travel search. We create a ranking of *tours* for a *set of travel interests*, where a tour is a group of city documents and a travel interest is a query. While generating and ranking tours, it is aimed that each interest (from the interest set) is satisfied by at least one city in a tour and the distance traveled to cover the tour is not too large. Firstly, we generate tours for the interest set, by utilizing the available ranking of cities for the individual interests and the distances between the cities. Then, in absence of existing methods directly related to our problem, we devise our novel techniques to calculate ranking scores for the tours and present a comparison of these techniques in our results. We demonstrate our web application *Travición*, that utilizes the best tour scoring technique.

CCS CONCEPTS

• **Information systems → Retrieval tasks and goals**;

KEYWORDS

Information Retrieval, Ranking, Search Result Organization

ACM Reference Format:
Mohamed Abdel Maksoud, Gaurav Pandey, and Shuaiqiang Wang. 2018. Finding Tours for a Set of Interests. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3184558.3186982

1 INTRODUCTION

In this paper, we present our novel method to address the problem of finding *tours* (or groups of cities) in response to a *set of travel interests*. When planning a trip, it is likely that a user desires her multiple travel interests to be catered. While it can be difficult to find many individual cities that satisfy a set of interests, the creation of relevant and practical tours is expected to enrich the offering to the user. We create ranking of tours in response to a set of interests, where each tour is a set of city documents and the set of interests is a combination of travel related queries (interests). To elaborate, for a set of interests $\{itr_1 \dots itr_n\}$ we create a ranking of tours $\langle t_1, \dots t_m \rangle$, where each t_i contains one or more cities. For this, we utilize the already available rankings of city documents for individual interests to form tours, while aiming that each interest in the interest set is satisfied by at least one city in the tour.

We see that extensive research has been carried out to cluster searched documents, in order to increase the coverage of results presented to the user [3, 4]. Also, there are many efforts for geo-spatial routing between locations [9, 14], rank aggregation [5] and group recommendation [12]. Moreover, there are recent efforts aimed to provide tours to users [2, 6, 7, 10]. To the best of our knowledge, there are no existing methods that address the tour creation problem in our way: i.e. a) by utilizing document rankings for individual queries, to generate *document groups* and rank them in response to a particular *set of queries*, b) while aiming that a top ranked document group satisfies all the queries in the set.

Our method initially generates the candidate tours and then uses novel scoring techniques to rank them. We utilize city rankings using our *CitySearcher* algorithm [1], that ranks cities (i.e. documents with information about one city each), in response to a travel interest (query). In response to a set of interests, we prune the city rankings for individual interests, to keep only the highly relevant cities. Then, we form the collection of candidate tours, so that each tour includes 0 or 1 city from each pruned city ranking for interests. Hence, for n interests, the candidate tours would be sets of cities of size 1 to n. We further filter out the candidate tours in which the cities are too far apart, since such tours are not practical. Since our method is unique and does not have related existing baselines, we rank the candidate tours using our novel and relatively simple scoring techniques, that use different combinations of a) available city-interest relevance scores and b) distance required to cover cities in a tour. On experimentation, the best scoring technique is identified and used in the *Travición* web-application: www.travicion.com, that we demonstrate.

Our presented web-application is intuitive and has an easy-to-use interface, that enables the user to select her set of travel interests and finds relevant tours for her.

2 METHOD

Here we describe objectives of *Travición* web-application, followed by tour generation, tour scoring techniques and implementation.

2.1 Objectives

Given: We have the following information available:

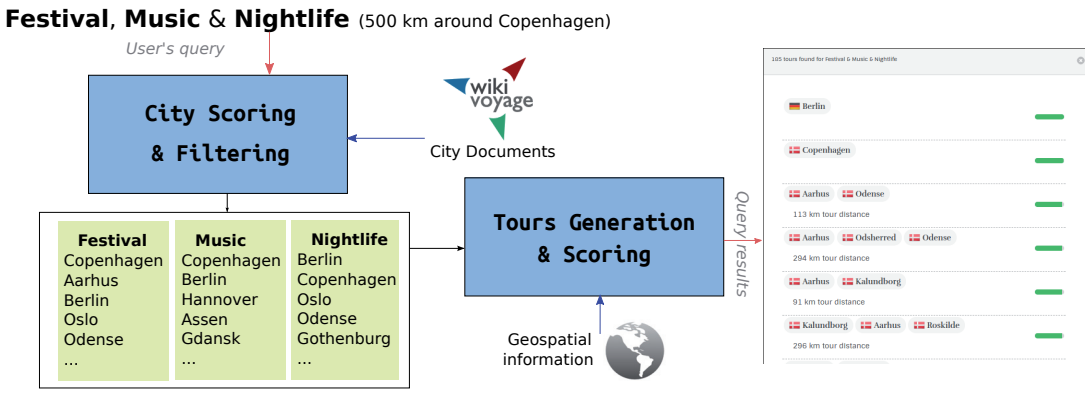
- A set of m documents representing different cities $\{c_1 \dots c_m\}$, having one to one mapping between cities and documents.
- A set of n travel interests $\{itr_1 \dots itr_n\}$ (provided by user), where each itr_i acts as query on city documents.
- A scoring function $score(c, itr)$ that gives a relevance score to a city c for an interest itr. This allows us to rank the m

*Indicates equal contribution

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3186982

Figure 1: Architecture of *Travición*



cities for each interest, i.e. for the interests $\{itr_1 \dots itr_n\}$ we have the corresponding city rankings: $\{cr_1 \dots cr_n\}$.

- A function $dist(C)$ that calculates the shortest geographical distance to travel cities $C \subset \{c_1 \dots c_m\}$, while starting and ending at any city in C.

Aim: For the set of interests $\{itr_1 \dots itr_n\}$, generate a single ranked list of tours i.e. $\langle t_1, \dots t_m \rangle$, where each tour t_j is a set of cities and $1 \leq |t_j| \leq n$.

Requirements: The ranking of tours should satisfy the following:

- Higher rankings should be assigned to those tours which address each interest, i.e. the tour contains cities, such that each interest is addressed by at least one city.
- Higher rankings should be assigned to those tours which contain cities that are not too distant from each other.

Remarks: It should be noted that we limit our scope and do not let the number of cities in a tour to be more than the number of interests, since one city is sufficient to address an interest. We admit that this would exclude some desirable tours containing two or more cities in close proximity and relevant to a particular interest.

2.2 Tour Generation

Given m cities and an an interest set with n interests, in order to create tours with 1 to n cities, there would be $\binom{m}{n} + \binom{m}{n-1} \dots \binom{m}{1}$ candidates, which could be a very large number. Therefore, for the n interests we generate the city rankings i.e. $\{cr_1 \dots cr_n\}$ (m cities each) [1], and prune them to keep only top k cities ($k << m$), as the cities occurring low in the city rankings of all the interests are quite unlikely to make good tours. Therefore, forming tours of size 1 to n, by taking 0 or 1 city from each of the n pruned city rankings with k cities each, there would be maximum $(k+1)^n - 1$ candidates. The set of candidate tours to be ranked can be denoted as $T = \{t_1 \dots t_p\}$, where $p \leq (k+1)^n - 1$ is the number of tours and each tour t_i is also a set of cities such that $0 < |t_i| \leq n$.

Moreover, as it can be expected that many of such tours would contain cities that are too far apart, we also put a threshold of D km on the pair-wise distance between cities in the same tour. Therefore, we filter out all such tours t_i from T where the distance between any pair of cities is greater than D. Note that we use pair-wise distance rather than the whole distance covered by the tour for performance reasons: organizing cities in a k-d tree [11] allows for efficient proximity search given a certain radius. This geospatial filtering further reduces the number of tours in T considerably.

2.3 Tour Scoring Techniques

We propose the following techniques to score a candidate tour t ($t \in T$) for the combined set of interests $I = \{itr_1 \dots itr_n\}$:

2.3.1 Distance-wise Scoring. A naïve approach could be to score a tour t as inverse of the distance covered in it i.e. $dist(t)$:

$$distScore(t, I) = \frac{1}{dist(t)} \tag{1}$$

The above scoring function would completely favor tours containing less cities, and is expected to rank single city tours at the top. Although the candidate tours consist of cities ranking high for interests, the function itself is independent of any relevance for interests.

2.3.2 Average Relevance Scoring. Since an interest is satisfied by even one city in a tour, we define the tour-interest relevance as the maximum of city-interest relevances for the cities in the tour, i.e.:

$$rel(t, itr) = max(score(ct_1, itr), \dots score(ct_x, itr)), \tag{2}$$

where $ct_1, \dots ct_x$ are the x cities contained in tour t. Using this, we calculate the score of a tour for an interest set $I = \{itr_1 \dots itr_n\}$, by calculating the average of relevances for individual interests as:

$$relScore_{avg}(t, I) = \frac{\sum_{i=1}^{n} rel(t, itr_i)}{n} \tag{3}$$

2.3.3 MaxMin Relevance Scoring. This technique is based on our intuition that it is important for a tour to have high relevance for *each* interest in the interest set I. If a tour is highly relevant for all interests in I except one, it would not satisfy the set I in entirety. To emphasize the impact of the interest to which the relevance of tour is the least, we define the following score:

$$\begin{aligned} relScore_{mm}(t, I) = max(rel(t, itr_1), \dots rel(t, itr_n)) \\ \times min(rel(t, itr_1), \dots rel(t, itr_n)) \end{aligned} \tag{4}$$

2.3.4 Hybrid Scoring. We see that Distance-wise scoring considers only the tour distance and ignores the relevance. On the other hand, Average Relevance and MaxMin Relevance scoring techniques, do not take into account the tour distances. Therefore, we combine them to derive two hybrid techniques, expecting that a combination would lead to improvement in tour ranking.

The first hybrid scoring combines Distance-wise Scoring and Average Relevance Scoring by combining the scores calculated in Equations 1 and 3:

$$hybScore_1(t,I) = \lambda_1\, distScore(t,I) + (1-\lambda_1)\, relScore_{avg}(t,I) \quad (5)$$

Similarly, second hybrid scoring combines Distance-wise Scoring and MaxMin Relevance Scoring by combining the scores calculated in Equations 1 and 4:

$$hybScore_2(t,I) = \lambda_2\, distScore(t,I) + (1-\lambda_2)\, relScore_{mm}(t,I), \quad (6)$$

where λ_1 and λ_2 are the weighting parameters used in combining the scores.

2.4 Implementation

Travición is implemented as a web-application and takes a set of interests as input from the user. It utilizes the English version of Wikivoyage[1] dataset containing 6691 documents, where predominantly each document is representative of a particular city. For deriving ranking of cities for individual interests along with the ranking scores, it uses CitySearcher approach [1].

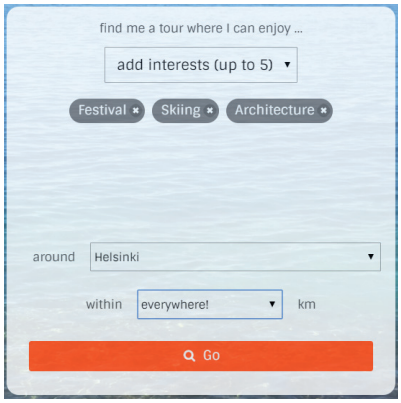
In the pruning step as described in Section 2.2, we limit the city ranking lists to their top 1000 results (i.e $k = 1000$). Distance between two cities is calculated by determining their respective latitude and longitude using the GeoLite dataset[2] and then using the Haversine formula [13]. Moreover, the threshold of pair-wise distances in one tour is set to 200 km (i.e. $D = 200$ in Section 2.2), so that the tours with one or more pair-wise distances more than D are not considered as candidates. Moreover, since in our evaluation (Section 4) $relScore_{mm}$ is the best performing technique, *Travición* employs it to score the tours. Additionally, the implementation also includes the functionality to exclude tours that are outside a specific region, i.e. the tours that lie outside a particular radius from a central location (this location and radius are optional inputs from the user). Figure 1 shows the architecture of *Travición* with the help of an example.

3 DEMONSTRATION SCENARIO

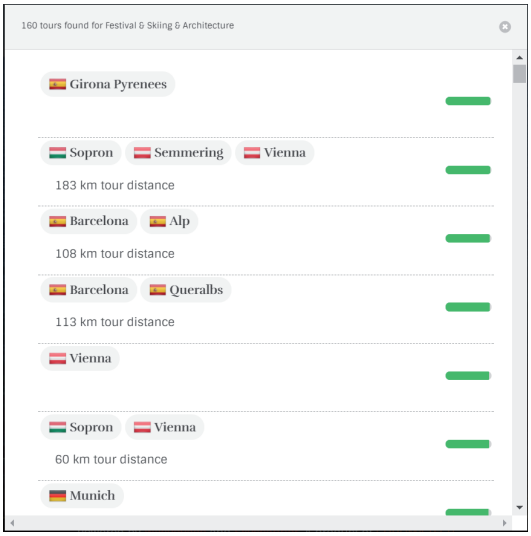
We plan to demonstrate our web application to the attendees, in which they can enter their travel interests to find a ranking of tours. Figure 2 shows the user interface of the *Travición* web application, which is quite intuitive. The user can choose up to 5 travel interests and press the 'Go' button. Thereafter, a ranked list of tours is presented to the user.

The search interface also allows the user to specify a specific region where they seek to do the tour. This region is specified by a center point (labeled 'around') and a radius in kilometers (labeled 'within'). Specifying a region effectively restricts the cities examined to those which are located in that region. A typical use

Figure 2: *Travición* User Interface



Tour Search



Search Results

case is when a user has a general idea where they want to travel (e.g. West Europe, East Asia), but they don't have specific destinations in mind. Therefore, she could choose a city in the mid of that region and specify the radius. Alternatively, if she wants to find tours around her current location, she shall choose it as the center point. Choosing 'everywhere' as radius would effectively consider all the world cities for tour formation. Hence, we demonstrate the scenario where a user specifies travel interests (and optionally a specific region) to discover a ranking of tours. The user shall expect that a high ranked tour would satisfy all her expressed interests (i.e. each interest should be addressed by at least one city in tour).

The ranking of tours contains single-city as well as multi-city tours as results. It should be noted that if the user selects only one interest, she would be presented with only single city tours. Moreover, if the user uses the 'everywhere!' option for the maximum distance from initial location, there are typically less multi-city tours in the top positions. This is because, with no restriction on

[1] https://en.wikivoyage.org/wiki/Main_Page
[2] https://dev.maxmind.com/geoip/geoip2/geolite2/

distance from initial location, many big cities around the world that can cater to multiple interests would rank high as tours.

On clicking a particular tour, the user is shown the tour on a map and then is also enabled to search for flights from an airport she specifies to one of the cities in the tour (not shown in the image). Flight and accommodation planning is not closely related to our demonstration or the problem we are presenting in this paper.
The web-application can be used on the web-page:
www.travicion.com
Also, an explanatory video / advertisement is available on:
www.youtube.com/watch?v=HlzmxSA2bjw

The hardware requirements for our demonstration are minimal, as the web-application only needs internet connection, and can be used on any modern web browser (on a computer or mobile device).

4 EXPERIMENTAL EVALUATION

For evaluation, we created a list of 50 travel interest sets, by choosing the most frequent travel interest sets requested on *Travición* website. Each interest set contains 3 or more interests and the average number of interests per interest set is 3.62. The experimental setup is same as in Section 2.4, except that we evaluate all the 5 scoring techniques ($distScore$, $relScore_{avg}$, $relScore_{mm}$, $hybScore_1$ and $hybScore_2$) described in Section 2.3. For this, we collect relevance assessments for top tours for these methods. We collected around 2500 ratings from 20 people using the crowdsourcing platform clickworker[3]. The ratings are taken on a scale of 1 to 20.

The standard ranking accuracy metric: normalized discounted cumulative gain (NDCG@1-5) [8] is used to compare our scoring techniques. Table 1 presents the experimental results for the scoring techniques: $distScore$, $relScore_{avg}$, $relScore_{mm}$, $hybScore_1$ and $hybScore_2$, formulated in Equations 1, 3, 4, 5 and 6 respectively. Weighting parameters λ_1 and λ_2 used in the scoring techniques $hybScore_1$ and $hybScore_2$ respectively, are assigned the value 0.5.

As expected, $distScore$ shows the worst results across the metrics, since it ranks the tours only on the basis of distance traveled within a tour. We see that $relScore_{mm}$ is the best performing technique and is significantly better than $relScore_{avg}$, confirming our intuition that the relevance of a tour to each interest in the interest set is more important than the average relevance for the interests.

Moreover, we see that the $hybScore_1$ (hybrid of $distScore$ and $relScore_{avg}$) shows slight improvement over $relScore_{avg}$. Also, $hybScore_2$ (hybrid of $distScore$ and $relScore_{mm}$) does not show any improvement over $relScore_{mm}$. Overall, the results of $relScore_{avg}$ and $relScore_{mm}$ do not differ a lot from their respective hybrids. Probably, this happens because the distances between the cities are already considered while filtering the candidate tours, the inclusion of tour distances in scoring is not very effective.

5 CONCLUSION AND FUTURE WORK

This paper demonstrates the *Travición* web-application that displays a ranking of tours in response to a set of travel interests. For this, we addressed the unique problem of creating and ranking document groups for a set of queries, so that each query is satisfied by at least one document in each group. We presented our novel techniques for ranking tours (groups of cities) for a set of travel interests, so that a tour satisfies all interests as well as cities in the tour are in geographical proximity. On comparison, we observe that the best results are achieved when a tour is scored using the product of its best and worst relevances for interests in the interest set.

In future, we plan to consider practical distances between cities within a tour, as we have used a rather naïve approach to calculate distances using Haversine formula. This is important especially when there are no direct roads between the cities, or there is a lake or mountain between them. Also, we aim to explore consideration of multiple cities per interest for tour formation.

[3] https://clickworker.com

Table 1: Comparison of Ranking Methods

NDCG	@1	@2	@3	@4	@5
distScore	0.6862	0.7012	0.6901	0.6964	0.6911
$relScore_{avg}$	0.7994	0.7958	0.7939	0.7951	0.7980
$relScore_{mm}$	**0.8662**	**0.8562**	**0.8526**	**0.8507**	**0.8472**
$hybScore_1$	0.7994	0.7958	0.7942	0.7954	0.7996
$hybScore_2$	0.8632	0.8532	0.8507	0.8494	0.8446

ACKNOWLEDGMENTS

This work was supported by Codoma.tech Advanced Technologies in the context of the *Travición* project (www.travicion.com) and the Academy of Finland (MineSocMed Grant No 268078).

REFERENCES

[1] Mohamed Abdel Maksoud, Gaurav Pandey, and Shuaiqiang Wang. 2017. CitySearcher: A City Search Engine For Interests. In *SIGIR*. 1141–1144.
[2] Flora Amato, Antonino Mazzeo, Vincenzo Moscato, and Antonio Picariello. 2014. Exploiting cloud technologies and context information for recommending touristic paths. In *Intelligent Distributed Computing VII*. Springer, 281–287.
[3] Claudio Carpineto, Stanislaw Osiński, Giovanni Romano, and Dawid Weiss. 2009. A Survey of Web Clustering Engines. *ACM Comput. Surv.* 41, 3 (2009), 17:1–17:38.
[4] Carlos Cobos, Henry Muñoz Collazos, Richar Urbano-Muñoz, Martha Mendoza, Elizabeth León, and Enrique Herrera-Viedma. 2014. Clustering of Web Search Results Based on the Cuckoo Search Algorithm and Balanced Bayesian Information Criterion. *Inf. Sci.* 281 (2014), 248–264.
[5] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. 2001. Rank Aggregation Methods for the Web. In *WWW*. 613–622.
[6] Laura Martinez Garcia, Jhonatan Montes Serna, and Valentina Codutti. 2017. 10 SIGNALS: A personalized city tours prototype. In *COLCOM*. 1–6.
[7] Damianos Gavalas, Michael Kenteris, Charalampos Konstantopoulos, and Grammati Pantziou. 2012. Web application for recommending personalised mobile tourist routes. *IET software* 6, 4 (2012), 313–322.
[8] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-based Evaluation of IR Techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.
[9] I. Johnson, J. Henderson, C. Perry, J. Schöning, and B. Hecht. 2017. Beautiful...but at What Cost?: An Examination of Externalities in Geographic Vehicle Routing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 2 (2017), 15:1–15:21.
[10] Kwan Hui Lim, Jeffrey Chan, Christopher Leckie, and Shanika Karunasekera. 2017. Personalized Trip Recommendation for Tourists based on User Interests, Points of Interest Visit Durations and Visit Recency. *Knowledge and Information Systems* (2017).
[11] Songrit Maneewongvatana and David M. Mount. 1999. Analysis of approximate nearest neighbor searching with clustered point sets. *CoRR* cs.CG/9901013 (1999).
[12] Judith Masthoff. 2015. Group Recommender Systems: Aggregation, Satisfaction and Group Attributes. In *Recommender Systems Handbook*. Springer, 743–776.
[13] Jovin J Mwemezi and Youfang Huang. 2011. Optimal Facility Location on Spherical Surfaces: Algorithm and Application. *New York Science Journal* 4, 7 (2011), 21–28.
[14] Jie Shao, Lars Kulik, Egemen Tanin, and Long Guo. 2014. Travel Distance Versus Navigation Complexity: A Study on Different Spatial Queries on Road Networks. In *CIKM*. 1791–1794.