USING LEARNING TO RANK FOR A TOP-N RECOMMENDATION SYSTEM IN TV DOMAIN

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BEDİA ACAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JUNE 2016

Approval of the thesis:

USING LEARNING TO RANK FOR A TOP-N RECOMMENDATION SYSTEM IN TV DOMAIN

submitted by **BEDİA ACAR** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Prof. Dr. Nihan Kesim Çiçekli _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Ali Doğru _____
Computer Engineering Dept., METU

Prof. Dr. Nihan Kesim Çiçekli _____
Computer Engineering Dept., METU

Asst. Prof. Selim Temizer _____
Computer Engineering Dept., METU

Asst. Prof. Çiğdem Turhan _____
Computer Engineering Dept., Atılım University

Asst. Prof. Gönenç Ercan _____
Computer Engineering Dept., Hacettepe University

Date: 29.06.2016

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: BEDİA ACAR

Signature :

ABSTRACT

USING LEARNING TO RANK FOR A TOP-N RECOMMENDATION SYSTEM IN TV DOMAIN

Acar, Bedia

M.Sc., Department of Computer Engineering

Supervisor: Prof. Dr. Nihan Kesim Çiçekli

June 2016, 59 pages

In this thesis, a top-N recommendation system in TV domain is proposed using learning to rank. The design, development and evaluation of the proposed recommender system are described in detail. Instead of calculating rating score of items like in conventional recommender systems, the ranked recommendation item list is presented to TV users. Moreover, path-based features which are used to build ranking model is explained in detail. These features provide collaborative filtering, content-based filtering and context aware recommendation system. Furthermore, some state of the art learning to rank approaches from each category called as pointwise, pairwise and listwise have been experimented to generate a ranking model. Then a baseline which does not use any learning are compared with the one using learning to rank algorithm. It is shown that the model constructed with learning to rank algorithm gives better results.

Keywords: Recommender systems, learning to rank, top-N recommendation, TV program recommendation, ranking

ÖZ

TV ALANINDA BİR İLK-N ÖNERİ SİSTEMİ İÇİN SIRALAMA ÖĞRENİMİNİN KULLANILMASI

Acar, Bedia

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Nihan Kesim Çiçekli

Haziran 2016, 59 sayfa

Bu çalışmada, sıralama öğrenimi yöntemleri kullanılarak televizyon alanında bir ilk-N öneri sistemi sunulmuştur. Önerilen sistemin tasarım, geliştirme ve değerlendirmesi detaylı bir şekilde açıklanmıştır. Geleneksel öneri sistemlerinde kullanılan nesnelerin puanlanmasının aksine, TV kullanıcılarına ilgiye göre sıralanmış öneri nesnelerinin listesi sunulmuştur. Bunun yanı sıra sıralama modelinin oluşturulması için kullanılan yol tabanlı özellikler detaylı bir şekilde açıklanmıştır. Kullanılan bu özellikler işbirlikçi, içerik tabanlı ve bağlam duyarlı öneri sistemini mümkün kılmaktadır. Ayrıca, bazı en gelişkin nokta bazlı, ikili bazlı ve liste bazlı olarak gruplandırılmış sıralama öğrenimi algoritmaları, sıralama modelinin oluşturulmasında denenmiştir. Böylece, temel bir öğrenme kullanmayan yöntem ile sıralama öğrenimi kullanan yöntem karşılaştırılmış ve sıralama öğrenimi kullanılan modelin daha iyi performans gösterdiği sunulmuştur.

Anahtar Kelimeler: Öneri sistemleri, sıralama öğrenimi, ilk-N öneri sistemleri, TV programı öneri sistemleri, sıralama

To My Family

ACKNOWLEDGMENTS

Firstly, I wish to express my sincere thanks to my supervisor Prof. Dr. Nihan Kesim Çiçekli for her valuable guidance, friendship and patience. This research would have never been accomplished without her motivation and illuminating views on this study.

I would like to thank Arda Taşçı for his helpful responses to my questions related to my thesis work and sharing his TV program data.

In addition, I would like to thank my colleagues in Tubitak Uzay for their support and encouragement through this thesis work. Working with them is also very valuable and is a big opportunity for me.

Finally, I am so grateful to my family for their love, encouragement and support through my life. Without their motivation, patience and prayers, it would be impossible to complete this thesis study.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vi

ACKNOWLEDGMENTS viii

LIST OF TABLES xi

LIST OF FIGURES xii

LIST OF ABBREVATIONS xiv

CHAPTERS 1

1. INTRODUCTION 1

1.1. Contribution of the Thesis 2

1.2. Organization of the Thesis 2

2. RELATED WORK 5

2.1. Recommender Systems in TV Domain 6

2.2. Learning to Rank 11

2.2.1. Learning to Rank Approaches 15

2.2.1.1. Pointwise Approach 15

2.2.1.2. Pairwise Approach 16

2.2.1.3. Listwise Approach 18

2.2.2. Data Labeling and Feature Extraction in Learning to Rank 19

2.3. Top-N Recommender Systems Using Learning to Rank 21

3. A TOP-N RECOMMENDATION FOR TV DOMAIN 25

3.1. Dataset Description 25

3.2. Construction of the Graph Based Model 27
3.2.1. Detailed Description of the Graph Based Model 29
3.3. Feature Extraction 33
3.3.1. Path Based Features 34
3.4. Ranking 37
3.4.1. Data Format of Training and Test File 39
3.4.1.1. Train Phase 40
3.4.1.2. Test Phase 41
4. EXPERIMENTS & EVALUATION 43
4.1. Evaluation Strategy and Metrics 43
4.1.1. K-fold Cross Validation 43
4.1.2. Evaluation Metrics 44
4.1.2.1. Recall @N 44
4.1.2.2. NDCG (Normalized Discounted Cumulative Gain) 45
4.2. Experimental Results 46
4.2.1. Comparison of Learning to Rank Algorithms 46
4.2.1.1. Recall @N Results 46
4.2.1.2. NDCG @N Results 48
4.2.2. Comparison with a Baseline Method 49
4.2.3. Comparison of Each Type of Path-Based Features 51
4.3. Discussion of Experimental Results 51
5. CONCLUSION 53
REFERENCES 55

LIST OF TABLES

Table 2.1 Counts of User-based and Item-based Features 21

Table 2.2 Weighted Frequency Values 22

Table 4.1 Learning to Rank Algorithms Compared 46

LIST OF FIGURES

Figure 2.1 Graph Based User Modeling..... 10

Figure 2.2 Machine Learning Framework (Liu 2011) 11

Figure 2.3 Typical flow of LTR applied search engine (Learning to Rank – Wikipedia, 2016)..... 12

Figure 2.4 Learning to Rank Framework (Liu 2011)..... 13

Figure 2.5 Weight Vectors w1 and w2 in Feature Space 17

Figure 3.1 A program instance in JSON program file 26

Figure 3.2 An Example Cypher Query..... 27

Figure 3.3 Sample Neo4j Graph..... 32

Figure 3.4 An Example Sub-graph..... 33

Figure 3.5 An Example Cypher Query to Find Path Counts..... 37

Figure 3.6 The Flow of Obtaining Top-N Recommendation List..... 38

Figure 3.7 Data Format 39

Figure 3.8 Sample Training File 40

Figure 3.9 Command to train data in RankLib..... 40

Figure 3.10 Command to train data for Ranking SVM..... 41

Figure 3.11 Command to test data in RankLib 41

Figure 3.12 Command to test data for RankingSVM..... 41

Figure 4.1 Five Fold Cross Validation 44

Figure 4.2 Recall @N values (N: 5, 10, 15 and 20) 47

Figure 4.3 Coordinate Ascent in 101 items..... 48

Figure 4.4 Comparison of 3 LTR methods according to NDCG @N..... 48

Figure 4.5 Comparison of Coordinate Ascent with SUM..... 49

Figure 4.6 Comparison of All Methods with a Baseline according to Recall @N . 50

Figure 4.7 Comparison of All Methods with a Baseline according to NDCG @N 50

Figure 4.8 Recall @N Results of Each Type of Path-Based Features 51

LIST OF ABBREVATIONS

API	Application Programming Interface
CBF	Content-based filtering
CF	Collaborative filtering
DCG	Discounted Cumulative Gain
EPG	Electronic Program Guide
IDF	Inverse Document Frequency
IR	Information Retrieval
JSON	JavaScript Object Notation
LTR	Learning To Rank
RS	Recommender System
SVM	Support Vector Machine
TF	Term Frequency
TV	Television
NDCG	Normalized Discounted Cumulative Gain

CHAPTER 1

INTRODUCTION

TVs have an important role in our daily life and Turkish people spend much of their time allocated for social activities in front of TV according to the research conducted by Turkish Statistical Institute (2015). Thus, it is inevitable that TV is the foremost entertainment tool for us. Moreover, the number of channels and programs are increasing day by day with the evolution of TVs. Today, there exist TVs which have the capability of Internet, such as Smart TV, IPTV and Internet TV. These different technologies also bring to access the Web sources besides serving traditional broadcasting media. As a result, user profiles can be constructed from the Web, social networks and TV watching history of users explicitly or implicitly (Véras, et al. 2015).

The necessity of recommender systems in TV domain has risen with the increase of the count of programs and channels. Instead of traditional channel surfing, recommending interesting and relevant contents will raise the satisfaction of TV users. Additionally, the availability of user profiles with the Internet connected TVs makes possible to develop recommender systems in TV domain.

Furthermore, the size of recommend item list must be reasonable since users will pay attention to only a few of them. In addition, the amount of recommended items, which can appear on the TV screen at a time, is very little. In this thesis, top-N item recommendation using learning to rank in TV domain is proposed. Firstly, path-based features in a graph based user model are extracted to generate a ranking model.

Then, according to this ranking model, top-N recommended item list is presented to the user.

1.1. Contribution of the Thesis

The present study contributes to the literature of recommendation systems in TV domain in the following ways;

- Path based features in the work of Ostuni, et al. (2013) are adapted to a graph based recommender system in TV domain. These features are extracted by using Neo4j.

- Some state of the art learning to rank approaches are applied to generate a ranking model and compared with each other. Then top-N items are recommended according to the ranking model.

- The used ranking model learns weights of features. It is compared with a baseline method which gives features equal weights. The evaluation results show that the ranking model performs better.

1.2. Organization of the Thesis

The rest of this thesis is organized as follows;

Chapter 2 summarizes the relevant literature on recommendation systems in TV domain by touching upon differences from other recommendation domains. Then, learning to rank (LTR) which is so popular in information retrieval domain is described in detail and the common types of LTR algorithms are presented. In the last sub-section of this chapter, some example recommendation systems which use learning to rank methods are explained.

Chapter 3 gives information about the developed system. Firstly, the graph based model is presented. Then, used path based features are described to construct the training data. Finally, the generation of the top-N recommended item list is described.

Chapter 4 illustrates the evaluation approaches used in LTR domain. Then, the conducted experiments and evaluation results are presented graphically.

Chapter 5 presents conclusions by summarizing the conducted work and suggesting relevant future work.

CHAPTER 2

RELATED WORK

Recommender Systems (RS) emerged in the mid-1990s as a solution to assist users by exploring items attracting them. In the last decades, recommendation has become a classic data mining problem which has been extensively researched in many application domains such as movie, book, music, news, shopping and others. To tackle this problem, the conventional recommender systems mostly try to predict the ratings of items. To accurately estimate the rating, core methodologies namely content-based filtering, collaborative filtering and hybrid of both have been developed (Lu, et al. 2015). Content-based filtering (CBF) is based on the similarity between descriptions of items and the profile of a user (Pazzani and Billsus 2007). On the other hand, the fundamental logic of collaborative filtering (CF) is that if n items are liked by two users, it means that the users have similar characteristics and thus they will rate or response other items similarly (Su and Khoshgoftaar 2009). Lastly, hybrid approaches combining two or more methods have also been proposed in the literature to increase the accuracy of predictions by taking advantages of each methods.

In this chapter, recommender system approaches in TV domain are presented by mentioning the methods and challenges faced in this domain. Moreover, top-N recommender systems and especially learning to rank, which is one of the mostly used approaches, are explained in detail.

2.1. Recommender Systems in TV Domain

Recommendation approaches have also been applied to TV domain since TV viewers suffer from finding interesting TV contents to watch among massive amount of TV programs and channels. Although research about TV program recommendation has been increased with the development of TVs and RSs, Véras, et al. state that more research must be done for TV programs since the recommendation for TV is more comprehensive and less studied than other recommendation domains (2015). The other point mentioned in their survey is that not only recommending TV programs, but also recommending other types of items is possible in TV domain. TV channels, advertisements, tourist packages, educational videos related with content can also be suggested to the user.

There exist some challenges in the recommendation of TV programs unlike other domains. One of them is the hidden user problem which means that multiple users, having different viewing habits and tastes, may share a single TV (Kim, et al. 2012). Therefore, different combinations of users in front of TV require different recommendations (Aharon, et al. 2015). Chang, Irvan and Terano state that TV program recommendation not only deals with personalized recommendation but also deals with group recommendation (2013). They also point that different members of a family can watch programs in distinct periods of day. For instance, the wife may prefer watching TV in the afternoon, children may prefer watching cartoon in the evening, and the husband may prefer watching news at night. At the weekends, family members may watch a program which is preferred by everyone, so group recommendation is required. An authentication system will also be cumbersome for family members in order to identify them explicitly.

Aharon, et al. developed a contextual TV recommender system namely *WatchItNext* which suggests programs to be watched by analyzing the context in which the current program is watched and time of day since currently watched program affects the next programs to be watched (2015). Thus, current context will give a clue about the target user profile to recommend a next program. They named this metric as sequential

popularity. The following recommender metrics are also defined in their study. In this thesis, general popularity and device popularity have also been considered by exploring paths on the graph.

- General Popularity; measures how many times the given item is watched for all devices.

- Temporal Popularity; measures the popularity of items in a particular time of day relative to its general popularity.

- Sequential Popularity; measures the popularity of items based upon their conditional popularity of being watched sequentially after a specific item.

- Device Popularity: measures the popularity of items based on how many times watched before by considering that users usually re-watch programs they have already watched.

Moreover, Aharon, et al. adapted Latent Dirichlet Allocation (LDA) algorithm as a collaborative filtering recommendation (2015). Latent Dirichlet Allocation is a learning algorithm to cluster words into latent topics and documents into mixtures of topics (Blei, Ng and Jordan 2003). To use LDA algorithm on TV data, every user (device) is considered as a document and every item watched by that user is considered as a word in the document. If the same program is watched multiple times, it will appear multiple times in a document or a user profile in that case. Each device may also be associated with multiple topics since many watchers who have varying characteristics, may share a single TV. After the calculation of the probability of each topic for a given user by LDA algorithm, the probability of an item to be watched for a given device (user) is estimated according to probability distribution of topics.

To tackle multiple users, Iguchi, Hijikata and Nishida try to identify users from logs according to the idea that there are one dominant user in different time intervals of a

day and in different days of a week and hence the active user has similar viewing patterns in his active times (2015). Then, they cluster days according to similar viewing patterns by running K-means algorithm and estimate the active time intervals for each cluster. Thus, a user profile is constructed according to each estimated active time in each cluster and the content of programs watched. As a result, the proposed model recommend items to the user according to time when he turns on TV. The relevance degree is the cosine similarity between user and program feature vectors. Each item of the feature vectors is TD-IDF score of a term which is extracted from texts in the EPG (Electronic Program Guide). TF-IDF (Term frequency – inverse document frequency) weighting schema is a popularly used metric in Informational Retrieval (IR).

In another study conducted by Kim, et al. users are modeled as feature vectors of category preferences where category is defined as the tuple of genre and channel (2012). Each element of the feature vector is calculated with CF-IUF (category frequency-inverse user frequency) scoring model adapted from TF-IDF weighting schema. Then users are clustered with ISOData algorithm which determines automatically the optimal number of clusters (Memarsadeghi, et al. 2007). After that, items are recommended based on the preference of similar users within the cluster by applying a user-based collaborative filtering approach. In the following equation, s_{uq} is the CF-IUF value of the user u for category q, f_{uq} is the number of views of category q, h_q is the number of users who viewed the category q and N is the total number of users. F_u is the feature vector of the user u consisting CF-IUF values for each category.

$$s_{uq} = \begin{cases} \left(1 + log f_{uq}\right).\ log \dfrac{N}{h_q}, & if\ f_{uq} \neq 0 \\ 0, & if\ f_{uq} = 0 \end{cases}$$

$$F_u = \{ s_{uq_1}, s_{uq_2}, \dots, s_{uq_m} \}$$

Oh, et al. states another significant point that not only recommended items but also time of the recommendation is significant in TV domain (2014). In conventional systems, user gets recommendation whatever time he wants. On the contrary, they developed a system which detects the proper time for recommendation while user watches TV. The proposed system continuously compare all programs broadcasted at that time with the user's currently watched one. When overall utility which is formulated in the below formula is positive, the item will be suggested to the user. Then user profile is adjusted according to the feedback of user.

$$Utility(u,t) = Profit(u,t) - Cost(u,t)$$

where *u* is a user and *t* is the time. While calculating the profit for each item, the system takes into consideration some measures such as how similar the item is to the user profile, how much time remains to the end of the program and how many times the user watches the same channel. The cost is calculated by taking into account the fact that user does not accept some recommended items consecutively.

The other challenge in TV domain is that collaborative filtering cannot advise newly broadcasted programs since none of the users has watched them yet. One solution is that programs can be considered as series, not as episodes. So if some episodes of an item are watched by similar users, we can suggest its subsequent episodes. Additionally, Taşçı proposes that collaborative filtering in TV domain can be applied in a fashion such that *"Users who are similar to user U are those who like the same content. Then, the user U is recommended those programs which are liked by the similar users"* (2015). Similar users are found by traversing the attribute and descriptor nodes through the graph based core model.

Taşçı proposes a graph based core model to represent users and programs (2015). The proposed model is enriched with content information of items (actor, director,

term and named entity), and contextual information (genre and time of day) besides user and program nodes (Figure 2.1). Therefore, content-based, collaborative filtering and context-aware recommendation systems get chance to be developed with the help of this model. Additionally, the user can make elimination on the recommended item list by explicitly choosing the favorite actor, director, genre etc.

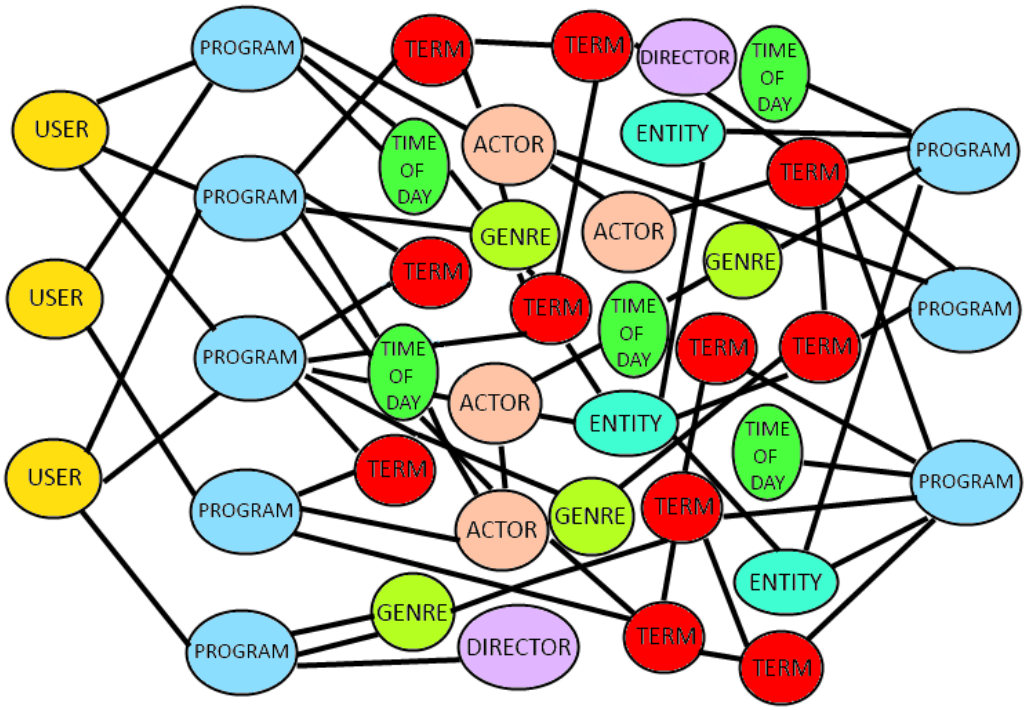


Figure 2.1 Graph Based User Modeling

In the study conducted by Taşçı, TV programs whether relevant or irrelevant to the user is determined by running spreading activation algorithm on graph-based model (2015). Spreading activation is an algorithm that labels nodes with weights by iteratively propagating through the network. These weights are values that decay while spreading through the network. At the end of this propagation, relevant items are found. In this study, a similar graph based core model is used in the representation of users and TV programs with their attributes. However, the primary goal of the

recommendation approach used in this thesis is to find top-N relevant programs in a ranked way instead of labeling programs as relevant or irrelevant. Moreover, the size of the recommended item list must be small to be able to show it on the TV screen and so it is significant to move up relevant items to the top positions of the recommendation list.

2.2. Learning to Rank

Supervised learning algorithms in machine learning contain 4 main components illustrated in Figure 2.2. These are input space, output space, hypothesis and loss function.

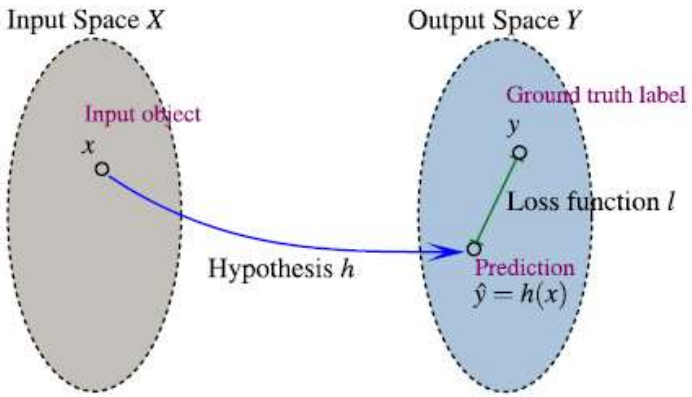


Figure 2.2 Machine Learning Framework (Liu 2011)

In brief, input space (X) is the input objects which will be used for learning the model. These input objects are usually represented as feature vectors. Output space (Y) is the target variable which will be predicted. While it takes discrete values in classification problems, it takes real values in regression problems. Unlike classification and regression problems, the output space is a sorted list of items in ranking problems. Furthermore, Liu states that two different definitions of output

space exist. While the first one is the output space of the task, the second one is the output space to facilitate learning. For instance, one can use regression algorithm for the classification problem. In that case, the output space of the task is the discrete labels while the output space to facilitate learning is the real numbers (2011). Hypothesis (h) is the function which maps input space to output space $h: X -> Y$. Lastly, the lost function (l) measures how much the predicted labels are similar to ground truth labels (y) which are given in training data.

Methods that use supervised machine learning techniques to solve the problem of ranking are distinctively called as learning to rank (LTR) (Liu 2011). Learning to rank algorithms learn a hypothesis function which outputs a score for each object. Additionally, an ordered list of objects are constructed by sorting these score values.

There exist applications using LTR in the fields of Information Retrieval (IR), Natural Language Processing and Data Mining etc. Some specific applications are web search, question answering, collaborative filtering, machine translation and text summarization. Recently, LTR is one of the key technologies for modern web search (Li 2011). The commercial web search engines have two ranking phases (Figure 2.3).

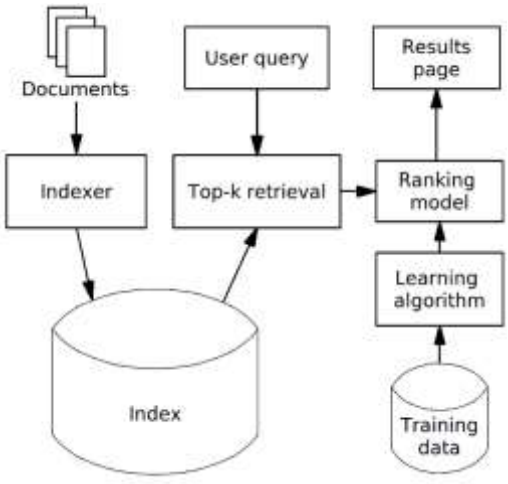


Figure 2.3 Typical flow of LTR applied search engine (Learning to Rank – Wikipedia, 2016)

Firstly, top-k documents are retrieved with applying traditional approaches, then top-k results are re-ranked according to the learned ranking model. The benchmark datasets are also available for the research on learning to rank for information retrieval, namely LETOR (Qin, et. al 2010).

Learning to rank will be first described from the information retrieval perspective since it is one of the most active research fields in IR. In information retrieval, the goal is to rank documents with respect to a given query. Figure 2.4 shows the steps in a learning to rank framework. Since LTR is a supervised learning task, it has two phases known as training and testing.

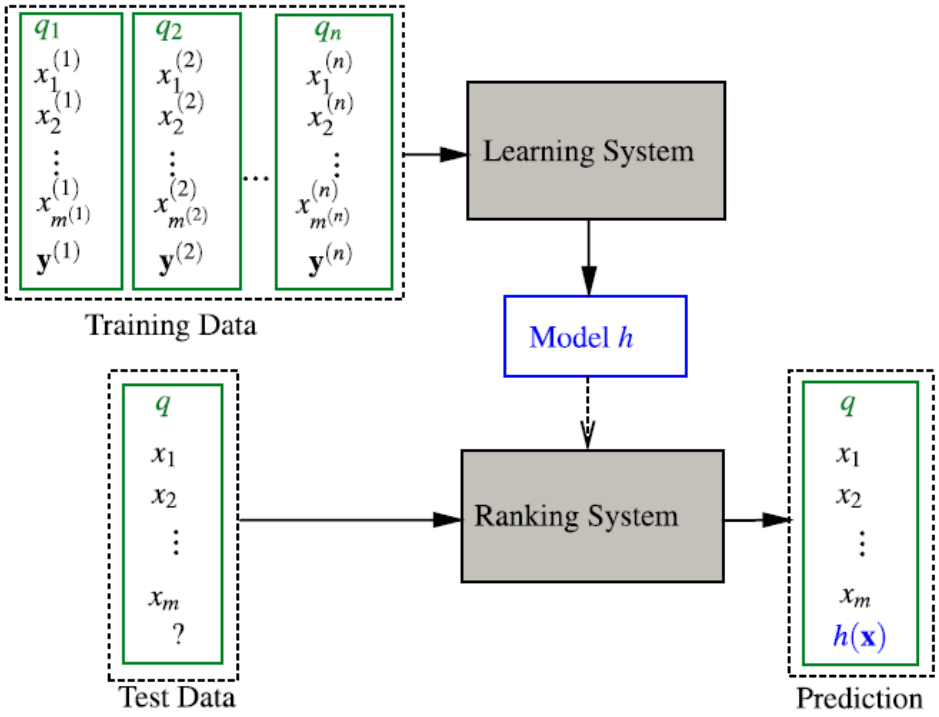


Figure 2.4 Learning to Rank Framework (Liu 2011)

In training phase, a ranking function $h(x) = f(q, d)$ is learned from the training data by applying one of the learning algorithms. Training data consists of n number

of queries q_i $(i = 1 \dots n)$ and m number of documents d_i $(i = 1 \dots m)$ where each of them is represented with a feature vector x_i, $x^{(i)} = \{x_j^{(i)}\}_{j=1}^{m^{(i)}}$. Additionally, each query and document pair is associated with a relevance label y (ground truth label). This label shows the degree of relevance between query and document.

In a similar way, test data consists of a query and a list of documents corresponding to query without a relevance label. The ranking system outputs a ranking list of documents for that query in the test phase according to the learned model h.

LTR methods have two main characteristics namely feature-based and discriminative training. (Liu 2011)

Feature based: It means that feature vectors are used in the representation of all documents. These features reveals the similarity of the documents with respect to the given query. Conventional systems make manual parameter tuning to predict the weight of each feature, which is so difficult especially when there are too many parameters. On the other hand, LTR endeavors to combine these features in an optimal way in order to predict the relevance label correctly.

Discriminative training: There exists an automatic learning process based on training data. Additionally, LTR methods can be described by four components of discriminative learning which are input space, output space, hypothesis space and loss function described in the beginning of the section. According to differences in these four components, learning to rank algorithms are classified into three common approaches namely, pointwise, pairwise and listwise. In LTR, the output space which facilitates learning is used to differentiate approaches from each other, but not the output space of the task.

2.2.1. Learning to Rank Approaches

2.2.1.1. Pointwise Approach

Pointwise approaches focus on the estimation of relevance score of each document separately. Thus, existing machine learning algorithms can be used to solve the problem of ranking. Since, pointwise algorithms do not care group structure, the problem of ranking can be reduced to classification, regression or ordinal regression. Therefore, the input space is a feature vector of each single document while the output space is the relevance degree of each single document. Furthermore, hypothesis space consists of a ranking function f which takes feature vectors of documents as input and outputs the relevance prediction of documents. Then, all documents are sorted to produce the final ranked list based upon labels returning from the function f. Regression loss, classification loss and ordinal regression loss functions can be used as a loss function since pointwise algorithms can be reduced to classification, regression or ordinal regression problem. Some pointwise algorithms are Random Forests (Breiman 2001), Gradient Boosted Regression Trees (Friedman 2001), McRank (Li, Burges and Wu 2008) and PRank (Crammer and Singer 2001). Mohan, Chen and Weinberger illustrate that Random Forests is comparable to or better than Gradient Boosted Regression Trees which is a standard algorithm for web search ranking. Therefore, Random Forests is evaluated among pointwise algorithms in this study.

Random Forests:

Random Forests for regression can also be used as a pointwise learning to rank algorithm. It is an ensemble method where a group of weak learners come together to form a strong learner. It consists of many decision trees, thus it is called forest. Each decision tree is constructed by selecting the random subset from the data. Then, the remaining data is tested on the tree and the out of bag error is calculated. After that, the tree with the lowest error is selected as the ranking model (Breiman 2001). Since, each tree is constructed independently from the others, this algorithm can

easily be parallelized. Finally, when a new object comes in the test phase, Random Forests averages the output of each decision trees.

2.2.1.2. Pairwise Approach

Unlike pointwise approaches, pairwise approaches take care of predicting the relative order between the pair of documents. Therefore, pairwise approaches try to maximize the number of correctly classified document pairs. Thus, the input space is pairs of documents where both of them are represented by feature vectors while the output space is the pairwise preference {+1, -1} showing which one is more relevant than the other. The hypothesis function takes a pair of documents and outputs the relative order between them. Then all documents are ranked according to ranking between pairs. If orders between all pairs are correctly sorted, then the optimal ranked list will be obtained. RankNet (Burges, et al. 2005), FRank (Tsai, et al. 2007), Ranking SVM (Joachims 2002) and RankBoost (Freund, et al. 2003) are some examples of pairwise methods. According to experiments conducted by Qin, et al., Ranking SVM gives slightly better results in some of the benchmark datasets compared to other pairwise algorithms namely FRank and RankBoost (2010). In this thesis, we have experimented Ranking SVM among pairwise approaches and it is introduced in the following part.

Ranking SVM:

SVM is a classification or regression analysis method which is widely used in various application areas. It can also be adapted for ranking problems and it is called as Ranking SVM. Differences between SVM for classification and Ranking SVM are as follows (Yu and Kim 2012);

- A training data is a set of objects with their class labels in classification; however, it is an ordered set of objects in Ranking SVM. Let the notation $A \succ B$ means that A is more relevant than B. Then a training set for ranking

SVM is $R = \{ (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ where y_i is the ranking of x_i, and if $x_i \succ x_j$, then $y_i < y_j$.

- The output of classification is the class label for an object. On the contrary, Ranking SVM outputs a score for each object such that $F(x_i) > F(x_j)$ for any $x_i \succ x_j$ where F is a ranking function. Then, a total ordering can be generated from these scores.

According to the following formula, Ranking SVM aims to learn a function F which finds a weight vector w such that $wx_i > w.x_j$ for most of the data pairs.

$$\forall\{(x_i, x_j): y_i < y_j \in R\}: F(x_j) \Longleftrightarrow wx_i > w.x_j$$

For instance; in the following Figure 2.5, weight vector $w1$ orders points as (1, 2, 3, 4, 5) and weight vector w2 orders points as (4, 1, 2, 5, 3) vector, so w2 cannot be a candidate vector.

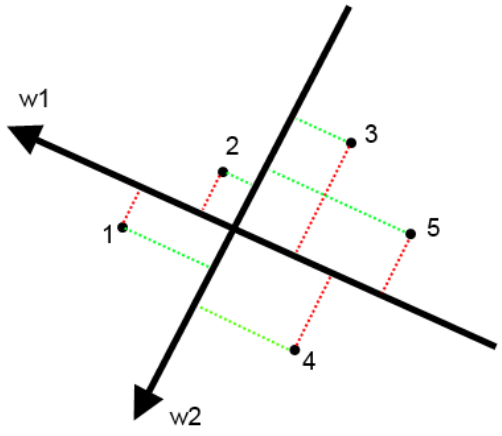


Figure 2.5 Weight Vectors w1 and w2 in Feature Space

Since finding such a vector w is NP-hard problem, we can find an approximate solution by using SVM methods with slack variables ξ_{ij} and minimizing the upper bound $\sum \xi_{ij}$ as the following:

$$minimize: \quad L_1(w, \xi_{ij}) = \frac{1}{2} w.w + C \sum \xi_{ij}$$
$$subject\ to: \forall\{(x_i, x_j): y_i < y_j \in R\} : w.x_i \geq w.x_j + 1 - \xi_{ij}$$

In the above formula, the part of $w.x_i \geq w.x_j + 1 - \xi_{ij}$ can be rewritten as;

$$w\,(x_i - x_j) \geq 1 - \xi_{ij}$$

Then this optimization problem becomes the classifying SVM on pairwise difference vectors $(x_i - x_j)$.

2.2.1.3. Listwise Approach

In listwise approach, a query and all documents associated with this query are given as input. Then, it outputs a permutation (ranked list) of the input documents such that the permutation minimizes the loss function. ListNet (Cao, et al. 2007), Coordinate Ascent (Metzler and Croft 2007), AdaRank (Xu and Li 2007), LambdaMART (Wu, et al. 2010) are some example algorithms applying listwise approach. In listwise approaches, lost function can also be considered as evaluation measure, so this approach also called as "direct optimization method" (Liu 2011). In the study of Busa-Fekete, et al., Coordinate Ascent gives the best performance with a small difference among some of the listwise algorithms such as AdaRank, LambdaMART and some of the pairwise algorithms such as RankBoost, RankingSVM (2012). As a listwise approach, Coordinate Ascent is chosen in this study.

Coordinate Ascent:

Coordinate Ascent is a listwise learning to rank approach proposed by Metzler and Croft (2007). It is a linear feature based method that optimizes the ranking measure. In this study, we have selected only one popular algorithm from each approach. These algorithms are Random Forests, Ranking SVM and Coordinate Ascent which have been already described above.

2.2.2. Data Labeling and Feature Extraction in Learning to Rank

Data Labeling

The training data given to learning algorithm consists of feature vector and relevance label for each user-item pair in recommender systems or document-query pairs in web search. Relevance labels can be inferred either explicitly or implicitly.

- *Explicit Feedback*

  Users explicitly clarify their interest on items by giving ratings, likes etc. In this way relevance labels can take binary or scalar values. In binary judgment, users decide whether the item is relevant or not, like or dislike. They give different degrees of scores to items such that perfect, excellent, good, fair and bad in scalar rating.

  Some drawbacks of explicit feedback are that;

  - Users may provide inconsistent or incorrect information.
  - Users' interest may change over time.
  - It is time consuming and a burden for users.

- *Implicit Feedback*

  Relevance labels can be inferred from logs or click-through information. The dwell time (i.e. how much time was spent) or the number of clicks on an item gives clue about the relevance degree of the item for the user. Since, explicit feedback is not possible in TV domain, implicit feedback has been used in this thesis. TV program watching time periods, which have been extracted from user logs, are used as relevance labels for constructing the training data.

Feature Extraction

Feature selection (or designing) is a significant phase to construct a good ranking model. Any attribute could be a feature if it is useful to the model. In machine learning, this is called feature engineering which requires the domain knowledge of data to make machine learning algorithms work.

In information retrieval area, page rank score of a document, title and length of the document, query length, TF-IDF score, occurrence counts of query terms in document etc. are some features that are commonly used (Qin, et. al 2010).

Learning to rank algorithms have also been applied to top-N recommendation systems in the last decades. Users are considered as queries and items are considered as documents to apply learning to rank in recommendation systems.

The following sub-section 2.3 explains how learning to rank is adapted to recommendation domain in detail by mentioning the types of features that have been used.

2.3. Top-N Recommender Systems Using Learning to Rank

In the last decades, approaches considering the recommender problem as ranking instead of rating have been emerged. Ranking based systems try to increase the fraction of the relevant items in the top n positions of the recommended item list. In particular, learning to rank (LTR) techniques which have demonstrated high promise, have been applied to generate top-N recommendations list recently (Shi, et al. 2012) (Weime, et al. 2009) (Sun, et al. 2012) (Ostuni, et al. 2013).

Sun, et al. adapted learning to rank for a hybrid movie recommendation system, namely LRHR (Learning to Rank for Hybrid Recommendation) for the first time (2012). For that, user and item based features are defined. The weights of these features are calculated with TF-IDF scores. Like documents and queries, users and items are represented with terms, namely user-based and item-based terms to make them content comparable. User-based terms are age interval, gender and occupation, item-based term is genre. In table 2.1, TF (term frequency) scores of each term for both user u and item i are illustrated. According to the table, user u is in the age interval of (18,24], male, technician, and u watched 64, 16 and 8 times of action, adventure, western movies respectively. Item i is watched by 10.000 people in the age interval of (18,24], 100.000 males, 1.000 technicians and the genre of item i is action and western.

Table 2.1 Counts of User-based and Item-based Features

	(18,24]	male	technician	student	action	adventure	western
User u	1	1	1	0	64	16	8
Item i	10.000	100.000	1.000	1.000	1	0	1

The frequency of terms have two weighting schema, TF and WF. User terms in the representation of users and item terms in the representation of items have low frequencies, generally which is 1, so TF is used in their weighting. However, other terms have high frequencies; therefore, WF weighting is used.

$$WF_{t,d} = \begin{cases} 1 + log_n TF_{t,d}, & if\ TF_{t,d} > 0 \\ 0, & otherwise \end{cases}$$

where *n* is 2 for users, 10 for items. New frequency values after weighting are shown in Table 2.2.

Table 2.2 Weighted Frequency Values

	(18,24]	male	technician	student	action	adventure	western
User u	1	1	1	0	7	5	4
Item i	5	6	4	4	1	0	1

Each user - item pair is defined with these term-based features and also some rating-based features. The relevance label of each user-item pair is the rating score [1 to 5] of an item given by the user. These feature vectors are input to Ranking SVM, pairwise learning to rank algorithm, in order to generate a ranking model.

In the study of Ostuni, et al., SPrank (Semantic Path-based ranking) algorithm is proposed (Ostuni, et al. 2013). Two dataset, MovieLens and Last.fm, which are widely used in recommendation domain, are chosen to test the proposed approach. They have used a graph based model, consisting user, item and named entity nodes. Named entities are extracted from DBPedia, a well-known encyclopedic knowledge base allowing users to query relationships and properties associated with Wikipedia

resources. From the analysis of the graph, they extract path-based features, assuming that the more paths between user and item, the more the item is relevant to user. Then, they used regression-based pointwise learning to rank algorithm to find which type of path features are most relevant to get a top-N recommendation list. In this thesis, similar path-based features are defined to generate a ranking model for recommendation in TV domain.

CHAPTER 3

A TOP-N RECOMMENDATION FOR TV DOMAIN

In this thesis, a top-N recommendation system for TV users has been developed by using learning to rank methods. The developed recommender system provides a hybrid recommendation framework using collaborative, content-based and context-aware features. In this chapter, the construction of graph based model, the feature extraction from the graph and the generation of ranked TV program list are described in detail.

3.1. Dataset Description

One month TV program dataset which was broadcasted in Turkish channels starting from 01.12.2013 and ending with 31.12.2013 has been used in this study. There are two types of data in the used dataset to construct the graph-based user model. The first one is the JSON files of programs, which are separated to different folders and files according to day and channel name. One JSON file encapsulates programs with their attributes which are name, date, start time, end time, genres, directors, actors, description, terms, and named-entities of a program for a given day and channel. Figure 3.1 illustrates one of the programs broadcasted on the day of 21.12.2013. The second type of data is user logs which are kept in MySQL database. Each record in this database consists of user id, start time, end time and channel name columns. The time interval between the start and end time shows the length of the watching activity

```
{
  "name": "Zengin Kız Fakir Oğlan",
  "date": "21.Aralık.2013",
  "startTimeStr": "04:10",
  "endTimeStr": "05:50",
  "duration": 100,
  "startTime": 1387591800,
  "endTime": 1387597800,
  "genreList": [
    "Dizi",
    "Komedi"
  ],
  "imageUrl":
"http://i.radikal.com.tr/TvRehberi/320x240/2013/12/21/5373780115201312210
410.jpg",
  "summary": "Zengin bir aileye damat giden genç bir adamın komik
öyküsü...",
  "longDescription": "Sıradan bir banka çalışanı olan Nurhan zengin bir aileye
damat olur. Kayınpeder Kemal ise onu kızına layık görmemektedir. Böylece
komik olaylar dizisi başlar.",
  "directors": [
    "Hasan Tolga Pulat"
  ],
  "actors": [
    "Hüseyin Avni Danyal",
    " Ayda Aksel",
    " Ufuk Özkan",
    " Ecem Özkaya"
  ],
  "stemmedWords": [
    "banka",
    "aile",
    "damat",
    "kemal",
    "onu",
    "kız",
    "layık",
    "olay",
    "dizi"
  ],
  "annotatedEntities": [],
  "timeOfDay": [ "EARLY_MORNING" ]
}
```


Figure 3.1 A program instance in JSON program file

of the user at a specified channel. Taşçı (2015) describes how these program attributes have been crawled and enhanced and how user logs have been collected in detail. In this thesis, a graph based model has been constructed thanks to these two types of data, the program JSON files and the user logs.

3.2. Construction of the Graph Based Model

Relational databases are not suitable for reflecting too many relationships; conversely, graph databases are proper for storing graph structures including many relationships. Therefore, Neo4j[1], which is one of the popular graph databases, has been chosen to construct a graph based user model in this study. Neo4j is a graph database management system implemented in Java. It stores everything as a node, an edge or an attribute. Each node and edge can have any number of attributes. These attributes can be a primitive type like string, numeric or boolean value, or a list which consists of any primitive types (The Neo4j Developer Manual v3.0. 2016). Neo4j has also its own query language, known as Cypher. It is a declarative language inspired from SQL and is used to describe patterns in graphs. An example cypher query in Figure 3.2 returns the top *10* most popular programs broadcasted between dates *07.12.2013* - *12.12.2013* which are corresponding to timestamp values *1386367200* and *1386885599* respectively.

```
Match (u:USER) - [r:user_program] - (p:PROGRAM)
where filter(x in r.startTimeStampList where x <> 0 and (x>1386367200 and
x<1386885599))
return p.name as name, count((r)) as popularity order by popularity DESC
limit 10;
```

Figure 3.2 An Example Cypher Query

[1] http://neo4j.com/

In this study, Java programming language was chosen for the implementation of building the graph and extracting features based on this graph. Cypher also provides a Java API which enables connecting to the database and writing queries with Java.

To construct the graph based user model, the available data, program data and user logs, have been aggregated by matching channel names and time attributes. The program data was initially inserted into Neo4j database by parsing the program JSON files. Programs with their attributes actor, director, genre, time of day, term and named-entity, are inserted into the graph database as distinct nodes. Then, users who watched Turkish channels are extracted from MySQL database. After that, programs which were watched by the users are found by intersecting watching time of the users with the broadcasting time of programs as well as comparing channel names.

In this aggregation phase, some preprocessing were conducted similar to the work of Taşçı (2015). The first one is that the redundant entries in the user log data are eliminated such that start time is bigger than end time or time interval is invalid. The second one was for correcting channel names because there were channel names which were written in different formats. These are mapped to the same format as the following;

> *"TRT-1 HD", "TRT 1 HD", "TRT1"* and *"TRT 1"* are mapped to *"TRT 1"*
> *"NTV Spor"* and *"NTV SPOR"* are mapped to "*NTV SPOR"* and so on.

As a result, the number of nodes in the final graph database are as follows;

- 2678 users
- 2124 programs
- 3964 terms
- 1202 named entities
- 35 genres
- 7 time of day

- 2685 actors
- 730 directors

3.2.1. Detailed Description of the Graph Based Model

In this sub-section, the defined types of nodes and edges are described. Similar terminology with the work of Taşçı is used (2015).

Node Types:

- *User Node*

  Users only have an id attribute since this dataset does not contain any other user related information.

- *Program Node*

  Programs have all attributes specified in the JSON file. The programs which were broadcasted many times are represented with a single program node which contains a list of start and end timestamps. Whether the user watched the program or not in the given time interval is determined according to this attribute while traversing the graph.

- *Genre Node*

  There are 35 genres defined in the system. Some of them are *"Sinema"*, *"Aktüalite"*, *"Yaşam"*, *"Hobi"*, *"Komedi"*, *"Belgesel"* and *"Tarihi"* etc. While inserting genre nodes, some mapping is also conducted. For instance; *"Eğlence Programı"* and *"Eğlence"* are inserted as one unique node with the name *"Eğlence"* since they refer to the same genre.

- *Time of Day Node*

  There are 7 types of *Time of Day* node, which are *"EARLY MORNING"*, "*BREAKFAST"*, *"LATE MORNING"*, *"DAYTIME"*, *"EVENING"*, *"PRIME TIME"* and *"NIGHT".*

- *Named-Entity Node*

  Programs may have zero or more named-entity nodes. Named-entities, which are extracted from DBPedia, can be the name of a famous person ("Türkan Şoray", "Barış Manço"), a location ("Türkiye", "Aydın") or a general term ("futbol","Anayasa Mahkemesi", "Hollywood", "bisiklet") etc. depending upon terms in the description field of TV programs.

- *Actor Node*

  Programs may have zero or more actor nodes.

- *Director Node*

  Programs may have zero or more director nodes.

- *Term Node*

  Terms were extracted from the description of program which are already available in the JSON file.

Edge Types:

- *"user_program"* is the relationship between user and program node. This edge exists if the user watched the TV program. *"user_program"* edge has the list of "dwellTime" attribute such as *dwellTimeList:* [*0.235, 0, 0, 0, 0, 0.887, 0, 0, 0, 0.376*]. This list shows that the program broadcasted 10 times in one month and the user watched the program 3 times with dwell times 0.235, 0.887 and 0.376 respectively. Dwell time shows the degree of relevance between user and program. It is calculated in the following formula.

$$dwell\ time = \frac{watching\ duration}{total\ duration\ of\ program}$$

The edge types defined below are all un-weighted edges.

- *"program_genre"* is the relationship between program and genre node. One program may have genres more than one.

- *"program_term"* is the relationship between program and term node.

- *"program_entity"* is the relationship between program and named-entity node.
- *"program_timeofday"* is the relationship between program and time of day node.
- *"program_actor"* is the relationship between program and actor node.

- *"program_director"* is the relationship between program and director node.

- *"actor_coocurance"* is the relationship between two actor nodes. If two actors are acted in the same program, there is a co-occurrence relationship between them.
- *"entity_coocurance"* is the relationship between two named-entity nodes. If two entities are linked to the same program, there is a co-occurrence relationship between them.

- *"term_coocurance"* is the relationship between two term nodes. If two terms exist in the same program, there is a co-occurrence relationship between them.

  In Figure 3.3, a sample Neo4j graph is illustrated. The graph shows the programs watched by the user "27971" and the attributes of those programs.

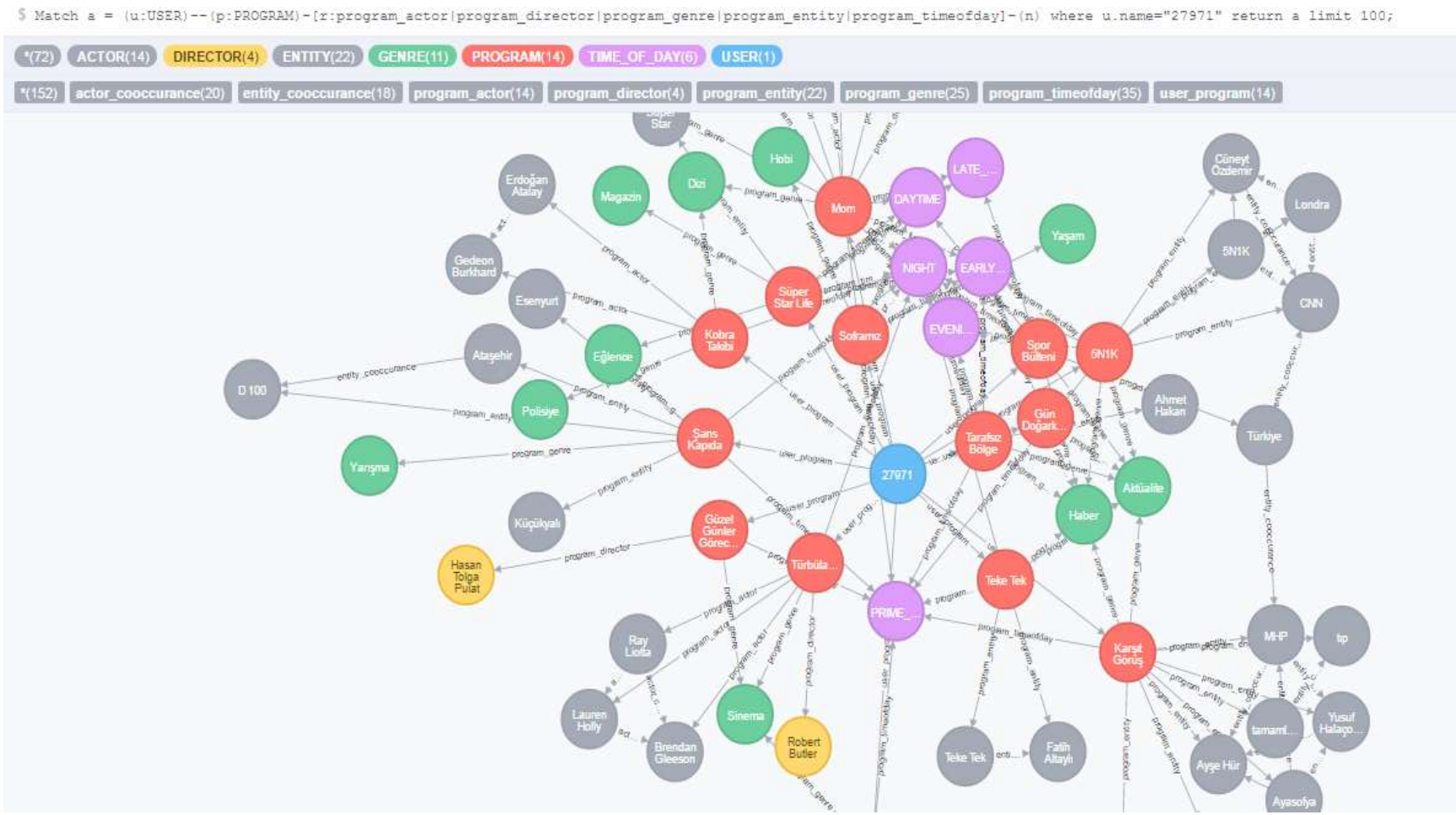


Figure 3.3 Sample Neo4j Graph

3.3. Feature Extraction

To generate a good ranking model, it is critical to define good features. SPRank (Semantic Path based Ranking) algorithm, which is proposed as a hybrid recommendation algorithm, uses path based features which are obtained from the graph based user model (Ostuni, et al. 2013). In this thesis, this algorithm has been adapted to the TV domain in a similar way. In their graph model only user, program and named-entity nodes are available. In addition to these nodes, genre, actor, director, time of day and term nodes are added into the constructed graph. The construction of path based features is explained with the help of the following sub-graph.

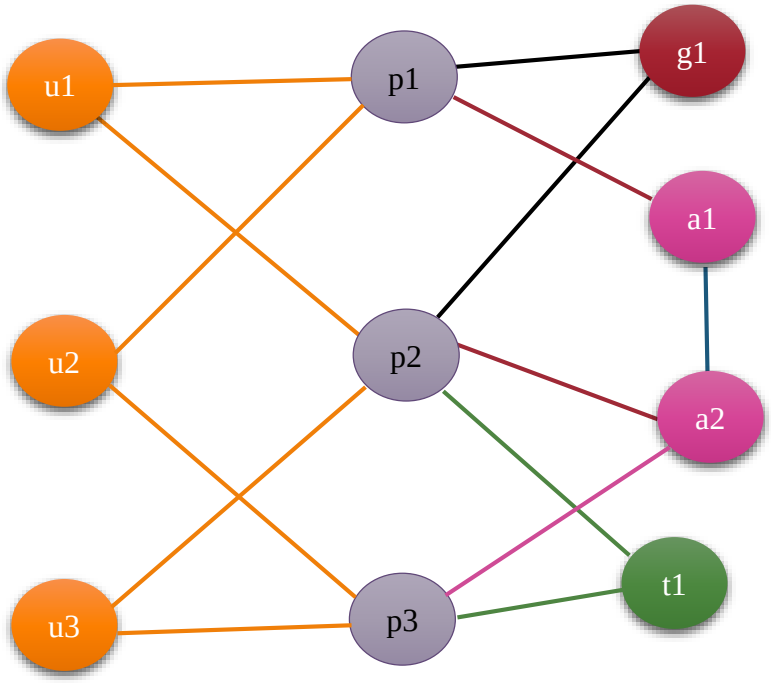


Figure 3.4 An Example Sub-graph

In the undirected sub-graph G = (V, R) that is illustrated in Figure 3.4, the defined vertices are V: u, p, g, a, t which represent user, program, genre, actor and term

respectively. All possible paths which are obtained by traversing this sub-graph can be categorized into 6 types by assuming the maximum depth of a path is four.

- P1 : u – p
- P2: u – p – u – p
- P3: u – p – t – p
- P4: u – p – g – p
- P5: u – p – a – p
- P6: u – p – a – a – p

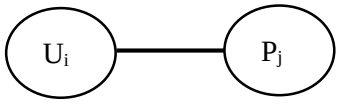
After defining these 6 path types, the number of paths for each kind of path is counted between the given user and item. For instance, if we choose the user u3 and the program p1, which has not been watched by the user u3 yet, we find all paths starting from u3 and ending with p1. There are two paths as *u3–p3–u2–p1* and *u3–p2–u1–p1* for the type of *u–p–u–p* path. There is the path of *u3–p2–g1–p1* for the type of *u–p–g–p*. Finally, there are paths of u3*–p3–a2–a1-p1* and *u3–p2–a2–a1-p1* for the type of *u–p–a–a–p.* So, the path counts are 0, 2, 0, 1, 0 and 2 for path types of P1, P2, P3, P4, P5 and P6 respectively.

3.3.1. Path Based Features

These paths are acyclic paths. The depth of paths is at most 4. These path based features can be grouped as rating based, collaborative, content based and context aware paths. The total path type is 11 and these 11 features are used to get the ranking model.

a) Rating-Based Paths

- User – Program



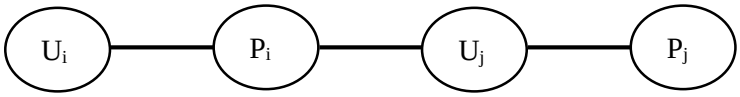
This type of path can be considered in two ways;

1) *For all users:* This will give how many users watched the program, which indicates the overall popularity of the program.

2) *For a specific user:* This will give how many times the user have watched the program.

b) Collaborative Paths

- User – Program – User – Program

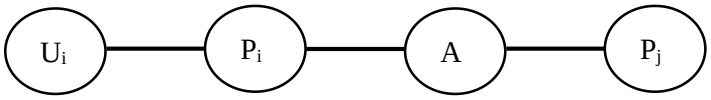


This path type indicates how many paths between the user node U_i and the program node P_j exist over any user node U_j who watched the same programs as the user U_i.

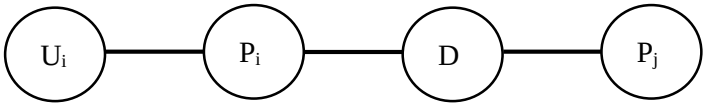
c) Content-based Paths

The followings are content-based paths that indicate the user U can watch programs consisting the same attributes with the already watched programs by U.

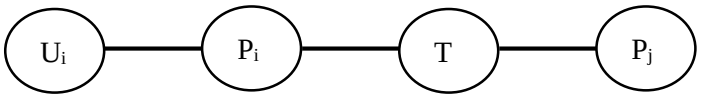
- User – Program – Actor – Program



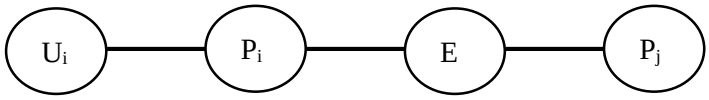
- User – Program – Director – Program



- User – Program – Term – Program

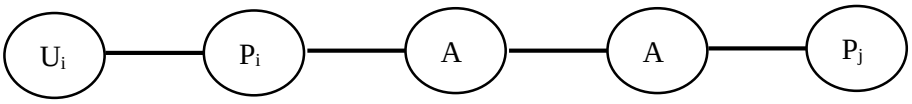


- User – Program – Named_Entity – Program

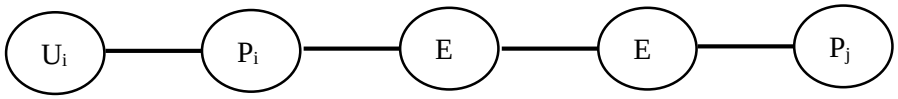


The following two types of path reveal the effect of co-occurrence in the actor-actor and entity-entity relationships.

- User – Program – Actor – Actor – Program



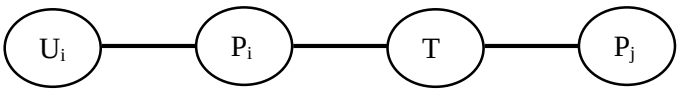
- User – Program – Named_Entity – Named_Entity – Program



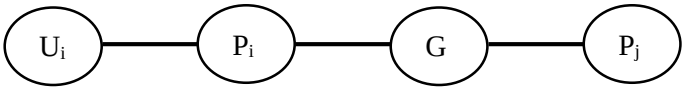
d) Context-Aware Paths

These paths consist of contextual nodes, namely time and genre.

- User – Program – TimeOfDay – Program



- User – Program – Genre – Program



An example cypher query returns the id of each program and all path counts between the user "27971" and each program node for the path type of "*user-program-actor-program"*.

```
MATCH path = (u:USER) -- (p1:PROGRAM) -- (a:ACTOR)-- (p2:PROGRAM)
WHERE u.name="27971"
RETURN ID(p2) as pid, count(path) as count
```

Figure 3.5 An Example Cypher Query to Find Path Counts

The weight of each kind of path features is different since some paths provide access to more relevant items while the importance of other paths may be less relevant (Ostuni, et al. 2013). Therefore, learning to rank provides a solution to enable an optimal ranking model according to the given features. In the following sub-section, we explain how learning to rank approaches have been used to get a ranked recommendation list.

3.4. Ranking

Figure 3.6 illustrates the flow of obtaining top-N recommendation program list. It consists of two phases, training and testing. First, a ranking model is learned from the training file. According to this model, predictions file generated for a user and all programs broadcasted in the specified time interval. In prediction file, there are score values for each program items. Then, these values are sorted to get a ranked list. After that, the top N program will be offered to the user.

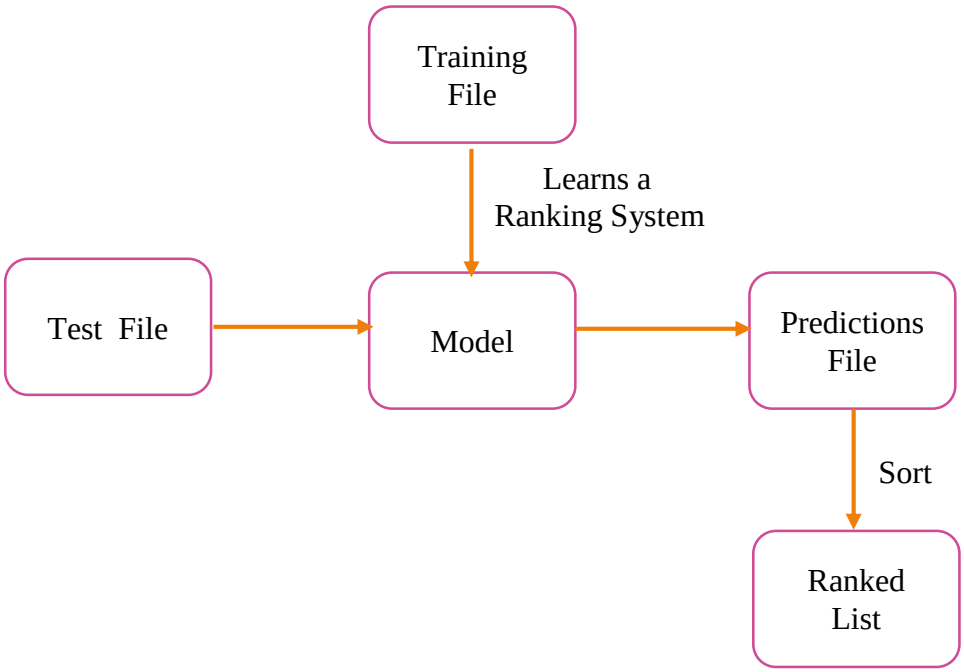


Figure 3.6 The Flow of Obtaining Top-N Recommendation List

To generate the ranking model, two learning to rank sources are used. The first one is RankLib[1] which is an open source library where some of the state of the art learning to rank algorithms implemented in Java. It is a part of Lemur project which supports the research and development of information retrieval and text mining by developing search engines, browser toolbars, text analysis tools, and data resources. From this library, Random Forests and Coordinate Ascent algorithms have been used. The second one is Ranking SVM[2] tool.

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

3.4.1. Data Format of Training and Test File

The data format is same for all algorithms which are used.

<line> .=. <relevance> qid:<qid> <feature>:<value> ... <feature>:<value>
<relevance> .=. 0 | 1
<qid> .=. <positive integer>
<feature> .=. <positive integer>
<value> .=. <float>

Figure 3.7 Data Format

Figure 3.7 illustrates a typical data format. Train and test file consist of lines where each line corresponds to one query and document pair. Each line starts with a relevance label and this label can take any integer number. In our case, relevance label is determined implicitly according to dwell time. In train and test phases, programs with dwell time bigger than 0.75 are considered as relevant, the others are considered as irrelevant. Therefore, relevance label takes only binary values, 0 or 1.

$$Relevance\ Label = \begin{cases} 1 & if\ \ dwellTime \geq 0.75 \\ 0 & otherwise \end{cases}$$

This relevance label is 0 in the test file since it will be predicted. After relevance label, the id of query comes and then *feature id: value* pairs (feature vectors) follow the query id. The feature ids must be in increasing order.

Figure 3.8 illustrates a sample training file. # is for commenting. In Figure 3.8, there are two queries with query id *(qid)* 1 and 2 and three documents returning for the both of these queries. There are also 6 features for all query document pairs.

```
1   qid:1  1:1 2:1 3:0 4:0.3 5:0  6:0.1  #1A
1   qid:1  1:1 2:0 3:1 4:0.5 5:1  6:0.2 #1B
0   qid:1  1:0 2:1 3:0 4:0.2 5:0   6:0.1#1C
0   qid:2  1:0 2:0 3:1 4:0.1 5:1 6:0.2  #2A
1   qid:2  1:1 2:1 3:1 4:0.3 5:1  6:0.2 #2B
0   qid:2  1:1 2:0 3:0 4:0.5 5:0 6:0.1  #2C
```

Figure 3.8 Sample Training File

3.4.1.1. Train Phase

It has been already mentioned that two different sources are used for learning to rank algorithms. For the first one RankLib, the command for training is shown in Figure 3.9.

```
> java -jar bin\RankLib.jar -train fold1\train.dat -ranker 4 -save
fold1\model
```

Figure 3.9 Command to train data in RankLib

Parameters

-train: specifies the training data file

-ranker: RankLib provides 9 learning to rank algorithms. This parameter shows which algorithm is used by specifying the id. Id is 4 for Coordinate Ascent, id is 8 for Random Forests.

-save: specifies the location of the learned model to save.

For the second tool RankingSVM, the command is as follows for training. It takes train file and model file as an argument.

```
> svm_rank_learn.exe train.dat model
```

Figure 3.10 Command to train data for Ranking SVM

3.4.1.2. Test Phase

For RankLib, the command for testing is as the following;

```
> java -jar bin\RankLib.jar -load fold1\model -rank fold1\fold1_test.dat
-score fold1\predictions
```

Figure 3.11 Command to test data in RankLib

Parameters:

-load: specifies which model file will be used.

-rank: specifies which test file will be used to get predictions file.

-score: specifies the location of predictions file.

For RankingSVM, the command to generate ranked list is as follows. It takes test file, model file and predictions file respectively as an argument.

```
> svm_rank_classify.exe test.dat model predictions
```

Figure 3.12 Command to test data for RankingSVM

CHAPTER 4

EXPERIMENTS & EVALUATION

The evaluation strategy and metrics used in the proposed recommendation system are explained in detail in this chapter. The metrics widely used in the evaluation of top-N recommendation systems are described and then the results of the conducted experiments are presented.

4.1. Evaluation Strategy and Metrics

4.1.1. K-fold Cross Validation

Cross validation, also known as rotation estimation, is a model validation technique which helps to observe the accuracy of model in practice. With the help of cross validation how the model will perform for unseen data is predicted. One commonly used kind of it is k-fold cross validation (Cross Validation, Wikipedia 2016). It starts with dividing the data into k chunks. One of the chunks is used for testing and remaining chunks are used for training. Then, it iterates k times until each chunk used as test data. Then result is calculated as the average of all rounds. Multiple rounds provide to reduce the variability.

In this work, 5-fold cross validation is chosen so data is divided into 5 chunks. For all evaluation results, 5-fold cross validation was performed. Since TV program data which had been broadcasted for one month, is used in this work, data is divided

according to equal time periods for evaluation (Figure 4.1). In test data, it is assumed that no user has watched any program yet.

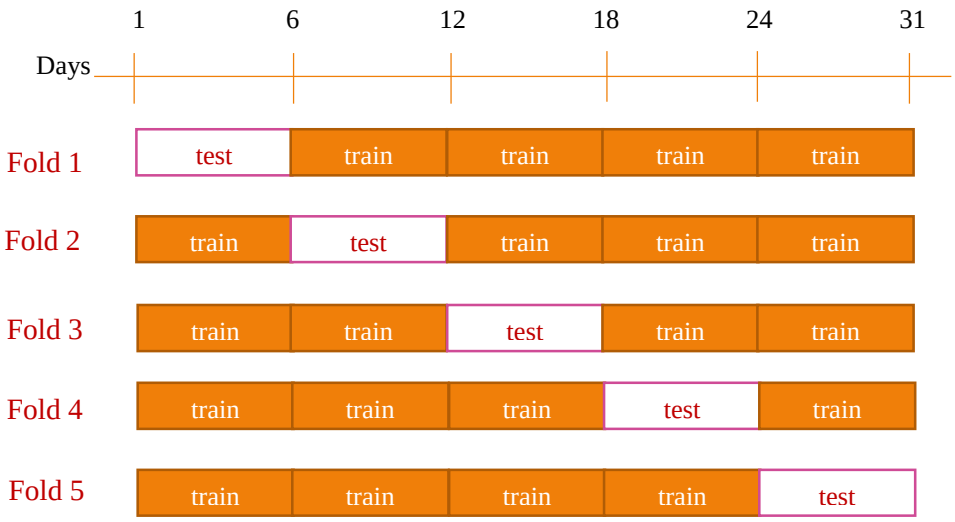


Figure 4.1 Five Fold Cross Validation

4.1.2. Evaluation Metrics

In top-N recommendation systems, there are some accepted measurements. Some of them are Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (NDCG), Recall @N and Mean Reciprocal Rank. In this study, Recall @N and NDCG have been used.

4.1.2.1. Recall @N

This metric measures how many items occur in the specified top-N positions of the recommended item list. The position of each relevant item i is compared with the value of given N. If the position is smaller than or equal to N, it is considered as **hit**, else it is considered as **miss**. Then total number of hits are averaged by total relevant items in the test data.

$$Recall@N = \frac{\#hits}{total\ relevant\ items}$$

In the calculation of Recall @N, two ways have been tried. The first one is looking at all items in test data. The second one is the same as the description in SPRank paper (Ostuni, et al. 2013). First, 100 irrelevant items are randomly selected from test data and only 1 relevant item is chosen. Then whether this 1 relevant item occurs at the specified top-N locations in these 101 items or not is evaluated.

4.1.2.2. NDCG (Normalized Discounted Cumulative Gain)

Discounted Cumulative Gain (DCG) measures the ranking quality, meaning that documents with higher label must be in top positions of the list. It penalizes the documents having higher grades if they are in lower positions of the list.

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)}$$

where p is the position and rel_i is the relevance value of item i.

NDCG is the fraction of DCG over IDCG (Ideal DCG).

$$NDCG_p = \frac{DCG_p}{IDCG_p}$$

For instance, assume there are documents d_1, d_2, d_3, d_4, d_5 and d_6 with values 3, 2, 3, 0, 1 and 2 respectively. Then DCG value of this ranking equals to 13.85. The optimal ordering of these documents are d_1, d_3, d_2, d_6, d_5 and d_4 with values 3, 3, 2, 2, 1, 0 and the value of IDCG is 14.6. So,

$$NDCG = \frac{13.85}{14.6} = 0.95$$

In this study, binary relevance judgment has been used and so rel_i in the NDCG formula, can take 0 or 1 value.

4.2. Experimental Results

In this thesis, three types of experiments are conducted by applying 5-fold cross validation technique. First, learning to rank algorithms are compared from each category. Second, a baseline method is compared against to learning to rank approaches. Finally, the effects of each type of path-based features are investigated. The results of each experiment are presented in the following sub-sections.

4.2.1. Comparison of Learning to Rank Algorithms

In evaluation, the state-of-the art learning to rank algorithms shown in Table 4.1 are tested. Each algorithm is chosen from different categories of learning to rank.

Table 4.1 Learning to Rank Algorithms Compared

Algorithm Name	**Category**
Random Forests	Pointwise
Ranking SVM	Pairwise
Coordinate Ascent	Listwise

4.2.1.1. Recall @N Results

As we mentioned in section 4.1.2.1 two approaches have been used for the calculation of Recall @N. Figure 4.2 shows the calculation of Recall @N with all

broadcasted items in test data. According to the results, Coordinate Ascent gives the best results with a small difference from Ranking SVM.

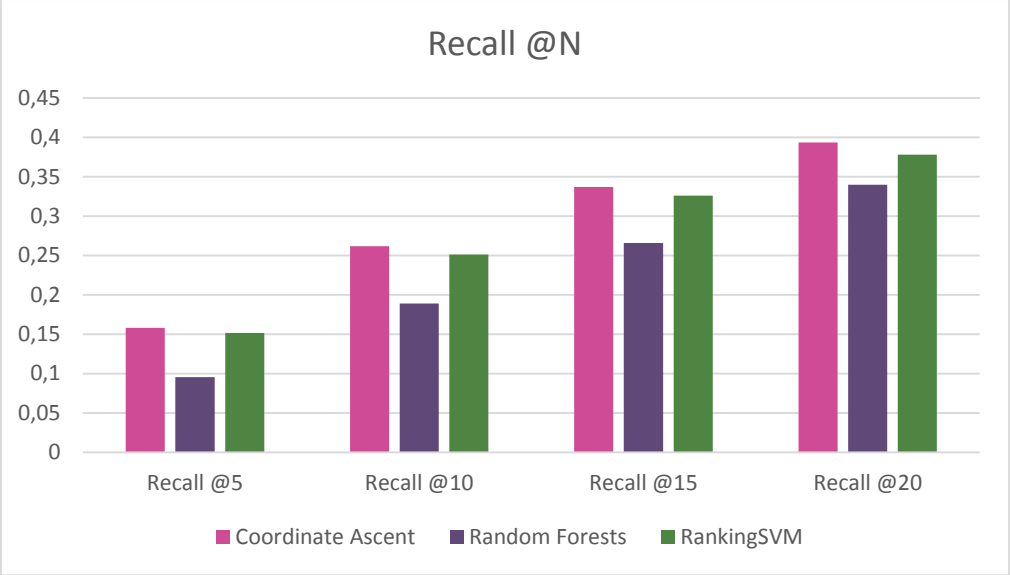


Figure 4.2 Recall @N values (N: 5, 10, 15 and 20)

Figure 4.3 illustrates the results of Coordinate Ascent (CA) for the second Recall @N approach that is selecting randomly 100 irrelevant items and 1 relevant item.

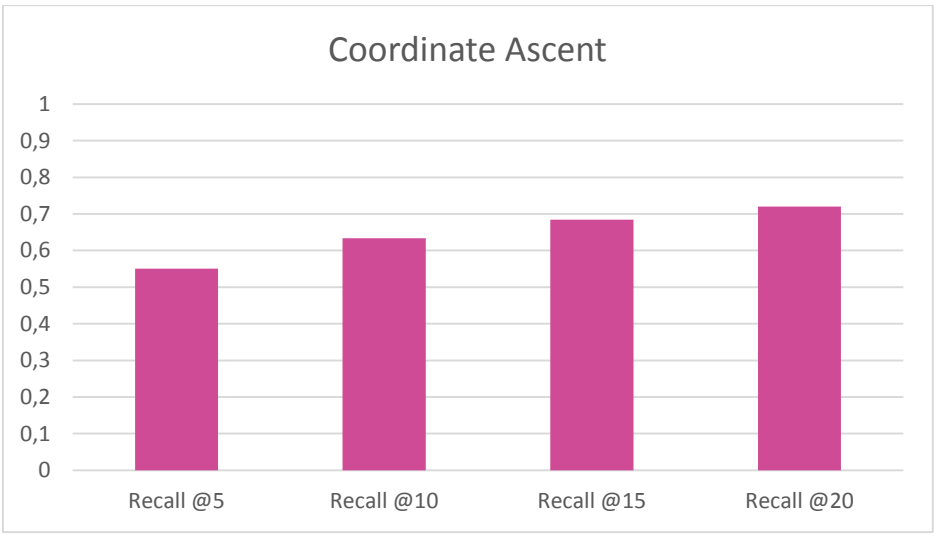


Figure 4.3 Coordinate Ascent in 101 items

4.2.1.2. NDCG @N Results

Results for three algorithms according to metric of NDCG @N are as the following;

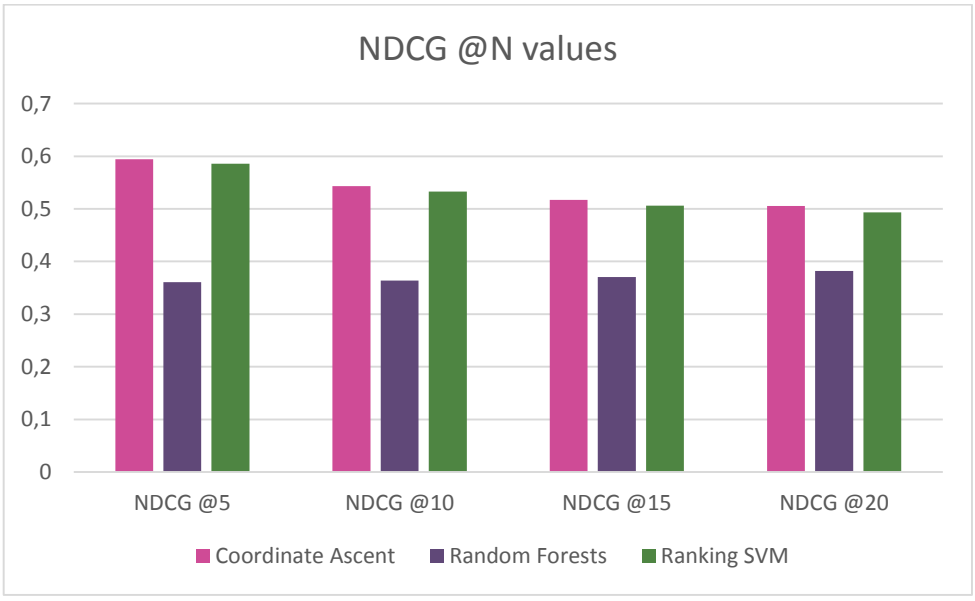


Figure 4.4 Comparison of 3 LTR methods according to NDCG @N

4.2.2. Comparison with a Baseline Method

In the work of Ostuni, et al. the proposed SPRank algorithm was compared with a baseline method namely Sum (2013). In Sum, all path count values for all features are summed up for each user and program. Therefore, a ranking-oriented learning approach is compared with a baseline method, which does not use any learning.

According to Figure 4.5, learning to rank approach, Coordinate Ascent outperforms Sum according to the evaluation metric Recall @N. In addition, Figure 4.6 compares all algorithms with the baseline method Sum. According to the presented results, all evaluated learning to rank approaches give better results than Sum except for Random Forests at Recall @5.

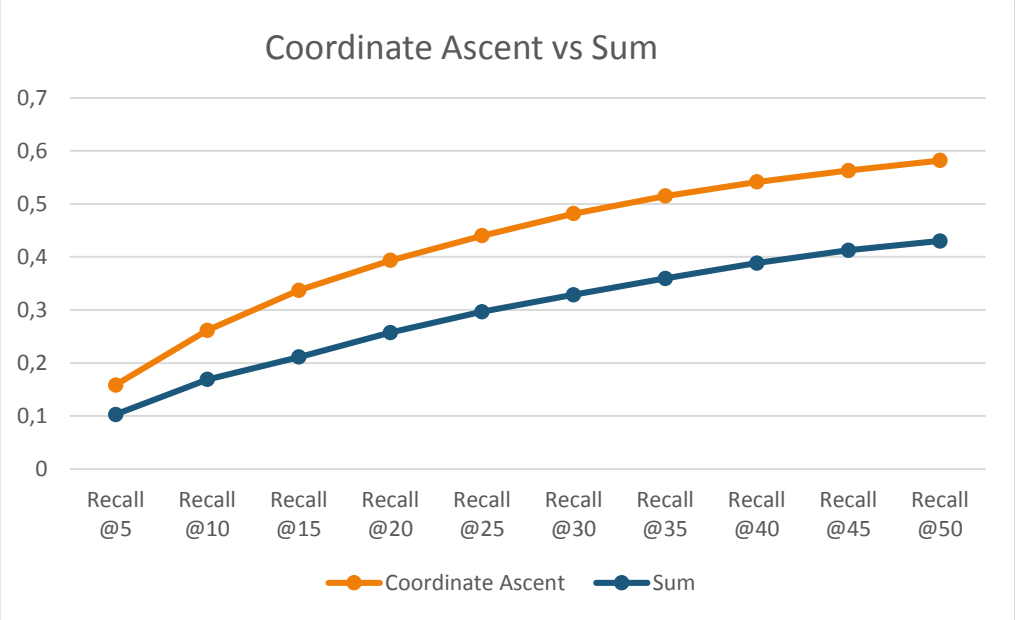


Figure 4.5 Comparison of Coordinate Ascent with SUM

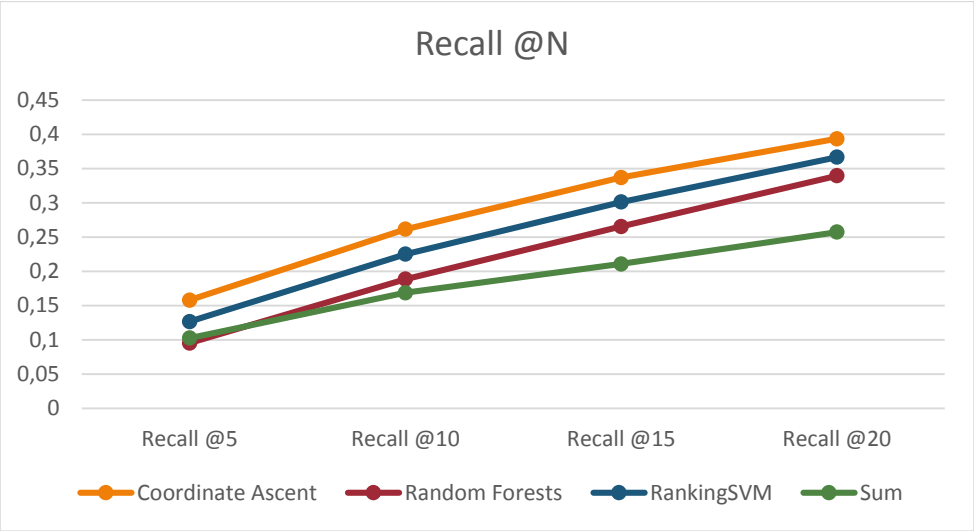


Figure 4.6 Comparison of All Methods with a Baseline according to Recall @N

Similarly, Figure 4.7 shows the comparison of all methods with Sum based on the metric of NDCG. As a result, learning to rank approaches give better NDCG@N values compared to Sum.

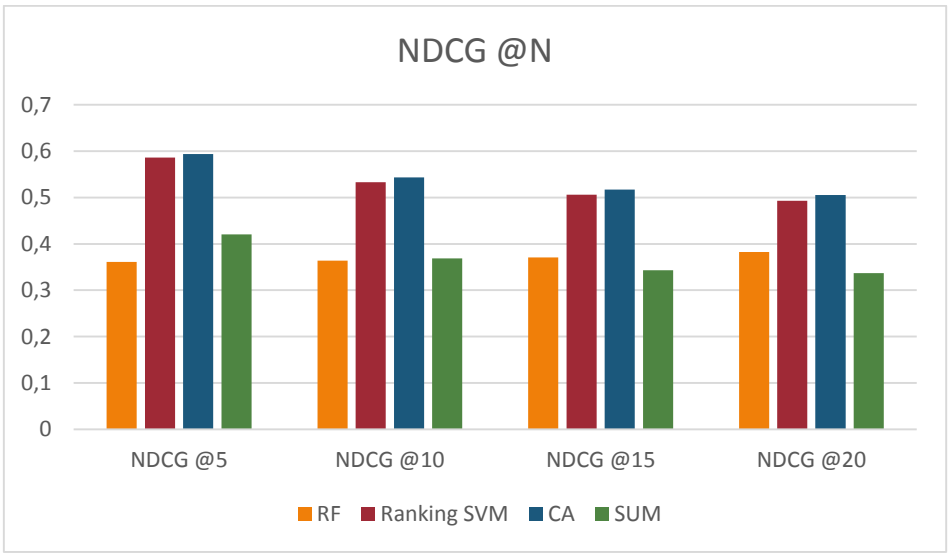


Figure 4.7 Comparison of All Methods with a Baseline according to NDCG @N

4.2.3. Comparison of Each Type of Path-Based Features

In this thesis, four types of path-based features are used. These are rating-based, collaborative, content-based and context-aware paths. We have experimented each type separately. Figure 4.8 illustrates the Recall @N results of each path type and also the Recall @N results of the overall. According to the results the most effective feature is collaborative path.

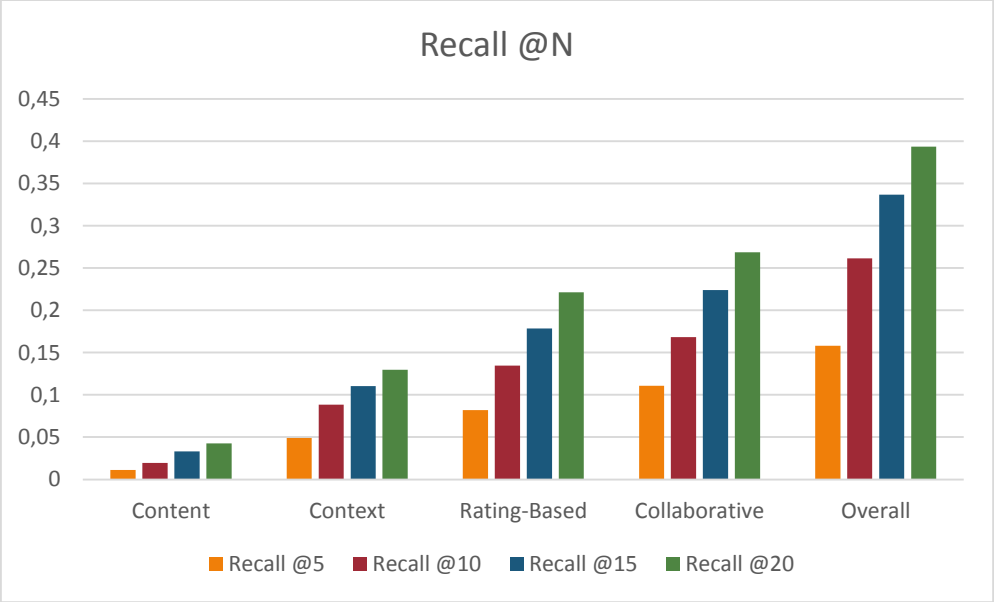


Figure 4.8 Recall @N Results of Each Type of Path-Based Features

4.3. Discussion of Experimental Results

In this thesis, it is mentioned that recommendation problem can be seen as a ranking problem instead of rating because suggested items to the users must be sorted according to the relevance. Thus, the users can find compelling programs near the top places of recommendation item list. In this study, we have developed a top-N recommender system in TV domain and we have considered only top-N locations of

the list where N equals to 5, 10, 15 and 20. The results show that learning to rank leads to move the relevant items to the top positions of the list. Furthermore, we have compared our recommender system with a baseline method in which all features have equal weights. According to the results, learning to rank approaches provide significant advantage compared to the baseline.

Moreover, we have compared three approaches of learning to rank. According to NDCG@N and Recall @N results, we can deduce that listwise approach (Coordinate Ascent) gives better results than pairwise and pointwise approaches as stated in the work of Qin, et al. (2010). The fact that listwise methods care the group structure of the list of program makes them advantageous relative to pointwise and pairwise methods. Moreover, Random Forests is slightly worse than Coordinate Ascent in this study as experimented by Busa-Fekete, et al. (2012).

Finally, we have investigated the effects of each type of path-based features. According to the results, we can conclude that the effect of collaborative filtering is the highest and then rating-based features follow it. The effect of the content is the lowest. The reason for that can be the sparseness of content-based nodes compared to the other nodes. For instance, many programs like whose genre are “News”, “Cartoon”, “Documentary”, “Sport” etc. do not have any connected actor, director nodes. On the other hand, each program has at least one genre and time-of-day node. Therefore, the effects of context can be observed more accurately.

CHAPTER 5

CONCLUSION

In this thesis, a top-N recommendation system has been developed by using the methods in learning to rank for TV domain. The main goal is to provide TV users a ranked list of programs for a specific top-N position.

First, users, programs and the attributes of programs are combined in a graph based model similar to the work of Taşçı (2015). Neo4j graph database management system was used both for the constructing of the graph based model and the acquiring of the features.

Moreover, features, which have an important role in the construction of a good ranking model, have been chosen. For that, path-based features, which have been proposed in the study of Ostuni, et al. (2013), were adapted to TV domain. Then, a ranking model was learned by using pointwise, pairwise and listwise LTR methods. These are, Random Forests, Ranking SVM and Coordinate Ascent respectively. Coordinate Ascent showed slightly better performance than RankingSVM. Furthermore, using learning to rank algorithms make an improvement compared to a baseline method, which aggregates all features equally.

As a future work, the count of features can be increased since learning to rank is so efficient to combine many features. Channel nodes can be added into the graph based user model, and then, channel name based ratings, such as how many times user watched the same channel and the general popularity of the channel can be used as

new features. Additionally, current context of the user is very significant in TV domain to recommend next programs to be watched as mentioned in the related work. So, two types of user profiles, namely current user profile and historical user profile, can be defined using the graph based user model. Then the same path-based features could be extracted from both current user profile and historical user profile to generate the ranking model.

REFERENCES

Aharon, Michal, Eshcar Hillel, Amit Kagian, Ronny Lempel, Hayim Makabee, and Raz Nissim. "Watch-It-Next: A Contextual TV Recommendation System." *Machine Learning and Knowledge Discovery in Databases Lecture Notes in Computer Science*, 2015: 180-195.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent Dirichlet Allocation." *Journal of Machine Learning Research*, 2003: 993-1022.

Breiman, Leo. "Random Forests." *Machine Learning* 45 (2001): 5-32.

Burges, Chris, et al. "Learning to Rank using Gradient Descent." *Proceedings of the 22nd International Conference on Machine Learning.* Bonn, 2005. 89-96.

Busa-Fekete, Robert, Gyorgy Szarvas, Tamas Elteto, and Balazs Kegl. "An apple-to-apple comparison of Learning-to-rank algorithms in terms of Normalized Discounted Cumulative." *In Proceedings of the 20th European Conference on Artificial Intelligence (ECAI-12)* 242 (2012).

Cao, Zhe, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. "Learning to Rank: From Pairwise Approach to Listwise Approach." *Proceedings of the 24th International Conferenceon Machine Learning.* 2007. 129-136.

Chang, Na, Mhd Irvan, and Takao Terano. "A TV program recommender framework." *Procedia Computer Science* 22 (2013): 561-570.

Crammer, Koby, and Yoram Singer. "Pranking with Ranking." *Advances in Neural Information Processing Systems*, 2001: 641-647.

Cremonesi, Paolo, Yehuda Koren, and Roberto Turrin. "Performance of Recommender Algorithms on Top-N Recommendation Tasks." *Proceedings of the fourth ACM conference on Recommender systems - RecSys '10.* 2010. 39-46.

Cross Validation - Wikipedia, the free encyclopedia. May 4, 2016. https://en.wikipedia.org/wiki/Cross-validation_(statistics) (accessed May 2016).

Discounted Cumulative Gain - Wikipedia, the free encyclopedia. April 11, 2016. https://en.wikipedia.org/wiki/Discounted_cumulative_gain (accessed May 2016).

Freund, Yoav, Raj Iyer, Robert E. Schapire, and Yoram Singer. "An Efficient Boosting Algorithm for Combining Preferences." *Journal of Machine Learning Research*, 2003: 933-969.

Friedman, Jerome H. "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics* 29 (2001): 1189-1232.

Iguchi, Koichi, Yoshinori Hijikata, and Shogo Nishida. "Individualizing user profile from viewing logs of several people for TV program recommendation." *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication - IMCOM '15.* BALI: ACM, 2015.

Joachims, Thorsten. "Optimizing Search Engines using Clickthrough Data." *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '02.* 2002. 133–142.

Kim, Myung-Won, Eun-Ju Kim, Won-Moon Song, Sung-Yeol Song, and A. Ra Khil. "Efficient Recommendation for Smart TV Contents." *Big Data Analytics Lecture Notes in Computer Science*, 2012: 158-167.

Learning to Rank - Wikipedia, the free encyclopedia. April 4, 2016. https://en.wikipedia.org/wiki/Learning_to_rank (accessed May 10, 2016).

Li, Hang. "A Short Introduction to Learning to Rank." *IEICE Transactions on Information and Systems IEICE Trans. Inf. & Syst.* E94-D, no. 10 (2011): 1854-1862.

Li, Ping, Christopher J.C. Burges, and Qiang Wu. "McRank: Learning to Rank Using Multiple Classification and Gradient Boosting." *Advances in Neural Information Processing Systems*, 2008: 845-852.

Liu, Tie-Yan. *Learning to Rank for Information Retrieval.* Heidelberg: Springer, 2011.

Lu, Jie, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. "Recommender system application developments: A survey." *Decision Support Systems* 74 (2015): 12-32.

Memarsadeghi, Nargess, David M. Mount, Nathan S. Netanyahu, and Jacqueline Le Moigne. "A Fast Implementation of the ISOData Clustering Algorithm." *International Journal of Computational Geometry & Applications*, 2007: 71-103.

Metzler, Donald, and W. Bruce Croft. "Linear Feature-Based Models for Information Retrieval." *Inf Retrieval*, 2007: 257-274.

Mohan, Ananth, Zheng Chen, and Kilian Weinberger. "Web-Search Ranking with Initialized Gradient Boosted Regression Trees." *Journal of Machine Learning Research*, 2011: 77-89.

Oh, Jinoh, Sungchul Kim, Jinha Kim, and Hwanjo Yu. "When to recommend: A new issue on TV show recommendation." *Information Sciences*, 2014: 261-274.

Ostuni, Vito Claudio, Tommaso Di Noia, Eugenio Di Sciascio, and Roberto Mirizzi. "Top-N Recommendations from Implicit Feedback Leveraging Linked Open Data." *Proceedings of the 7th ACM conference on Recommender systems - RecSys '13.* Hong Kong, China, 2013. 85-92.

Pazzani, Michael J., and Daniel Billsus. "Content-based Recommendation Systems." *The Adaptive Web Lecture Notes in Computer Science* 4321 (2007): 325-341.

Qin, Tao, Tie-Yan Liu, Jun Xu, and Hang Li. "LETOR: A benchmark collection for research on learning to rank for information retrieval." *Inf Retrieval* 13 (2010): 346-374.

Shi, Yue, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. "CLiMF: Collaborative Less-Is-More Filtering." *Proceedings of the sixth ACM conference on Recommender systems.* Dublin, 2012. 3077-3081.

Spreading Activation - Wikipedia, the free encyclopedia. September 22, 2015. https://en.wikipedia.org/wiki/Spreading_activation (accessed April 27, 2016).

Su, Xiaoyuan, and Taghi M. Khoshgoftaar. "A Survey of Collaborative Filtering Techniques." *Advances in Artificial Intelligence,* 2009: 1-19.

Sun, Jiankai, Shuaiqiang Wang, Byron J. Gao, and Jun Ma. "Learning to Rank for Hybrid Recommendation." *Proceedings of the 21st ACM international conference on Information and knowledge management - CIKM '12.* 2012. 2239-2242.

Taşçı, Arda. "A GRAPH-BASED CORE MODEL AND A HYBRID RECOMMENDER SYSTEM FOR TV USERS." MS Thesis, Middle East Technical University, 2015.

The Neo4j Developer Manual V3.0. 2016. http://neo4j.com/docs/developer-manual/current/ (accessed February 10, 2016).

Tsai, Ming-Feng, Tie-Yan Liu, Tao Qin, Hsin-Hsi Chen, and Wei-Ying Ma. "FRank: A Ranking Method with Fidelity Loss." *SIGIR '07.* New York: ACM Press, 2007. 383-390.

Tsunoda, Tomohiro, and Masaaki Hoshino. "Automatic metadata expansion and indirect collaborative filtering for TV program recommendation system." *Multimed Tools Appl* 36 (2008): 37-54.

Türkiye İstatistik Kurumu, Zaman Kullanım Araştırması. 2015. http://www.tuik.gov.tr/PreHaberBultenleri.do?id=18627 (accessed May 2016).

Véras, Douglas, Thiago Prota, Alysson Bispo, Ricardo Prudêncio, and Carlos Ferraz. "A literature review of recommender systems in the television domain." *Expert Systems With Applications* 42, no. 22 (2015): 9036-9046.

Weime, Markus, Alexandros Karatzoglou, Quoc Viet Le, and Alex Smola. "Cofi Rank Maximum Margin Matrix Factorization for Collaborative Filtering." *In Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems.* 2009.

Wu, Qiang, Christopher J. C. Burges, Krysta M. Svore, and Jianfeng Gao. "Adapting Boosting for Information Retrieval Measures." *Information Retrieval*, 2010: 254-270.

Xu, Jun, and Hang Li. "AdaRank: A Boosting Algorithm for Information Retrieval." *Proceedings of the 30th Annual Internation aCM SIGIR Conference on Research and Development in Information Retrieval.* 2007. 391-398.

Yu, Hwanjo, and Sungchul Kim. "SVM Tutorial : Classification, Regression and Ranking." *Handbook of Natural Computing*, 2012: 479-506.