

Institutionen för datavetenskap
Department of Computer and Information Science

Master's Thesis

**A Comparison of Katz-eig and Link-analysis for
Implicit Feedback Recommender Systems**

Jonas Hietala

LIU-IDA/LITH-EX-A-15/026-SE
Linköping 2015



Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Institutionen för datavetenskap

Department of Computer and Information Science

Master's Thesis

A Comparison of Katz-eig and Link-analysis for Implicit Feedback Recommender Systems


Jonas Hietala

LIU-IDA/LITH-EX-A-15/026-SE
Linköping 2015

Supervisor: **Mattias Tiger**
IDA, Linköpings universitet
Niklas Ekvall
Comordo Technologies

Examiner: **Fredrik Heintz**
IDA, Linköpings universitet

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

	Avdelning, Institution Division, Department AIICS Department of Computer and Information Science SE-581 83 Linköping	Datum Date 2015-06-10
Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____	Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	ISBN _____ ISRN LIU-IDA/LITH-EX-A-15/026-SE Serietitel och serienummer Title of series, numbering _____ ISSN _____
URL för elektronisk version http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-119169		
Titel Title En jämförelse av Katz-eig och Link-analysis för rekommendationssystem med implicit återkoppling A Comparison of Katz-eig and Link-analysis for Implicit Feedback Recommender Systems Författare Author Jonas Hietala		
Sammanfattning Abstract <p>Recommendations are becoming more and more important in a world where there is an abundance of possible choices and e-commerce and content providers are featuring recommendations prominently. Recommendations based on <i>explicit feedback</i>, where user is giving feedback for example with ratings, has been a popular research subject. <i>Implicit feedback</i> recommender systems which passively collects information about the users is an area growing in interest. It makes it possible to generate recommendations based purely from a user's interactions history without requiring any explicit input from the users, which is commercially useful for a wide area of businesses. This thesis builds a recommender system based on implicit feedback using the recommendation algorithms <i>katz-eig</i> and <i>link-analysis</i> and analyzes and implements strategies for learning optimized parameters for different datasets. The resulting system forms the foundation for Comordo Technologies' commercial recommender system.</p>		
Nyckelord Keywords katz-eig, link analysis, recommendations, machine learning		

Abstract

Recommendations are becoming more and more important in a world where there is an abundance of possible choices and e-commerce and content providers are featuring recommendations prominently. Recommendations based on *explicit feedback*, where user is giving feedback for example with ratings, has been a popular research subject. *Implicit feedback* recommender systems which passively collects information about the users is an area growing in interest. It makes it possible to generate recommendations based purely from a user's interactions history without requiring any explicit input from the users, which is commercially useful for a wide area of businesses. This thesis builds a recommender system based on implicit feedback using the recommendation algorithms *katz-eig* and *link-analysis* and analyzes and implements strategies for learning optimized parameters for different datasets. The resulting system forms the foundation for Comordo Technologies' commercial recommender system.

Acknowledgments

All thanks to Veronica who has been a pillar and a saint during these laborous times. Also big thanks to my supervisor Mattias Tiger who helped me write this thesis and to Niklas Ekwall and Comordo Technologies for support and for giving me the opportunity for this thesis work. Also thanks to my friend and opponent James Li who helped me improve my work.

Linköping, June 2015
Jonas Hietala

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Problem definition	2
1.2.1	Guiding questions	3
1.3	Limitations	3
1.4	Contributions	3
1.5	Outline of the report	4
2	Background	5
2.1	Recommendation theory	5
2.1.1	Recommendation model	6
2.1.2	Recommendation prediction	7
2.1.3	The katz-eig algorithm	8
2.1.4	The link-analysis algorithm	11
2.2	Machine learning	16
2.2.1	Supervised learning	16
2.2.2	Unsupervised learning	17
2.2.3	Evaluation	17
2.3	Optimization	19
3	Related work	21
4	The Comordo recommender system	23
4.1	Comordo	23
4.2	System development task	24
4.2.1	Use case	25
4.3	Development methodology	25
4.3.1	Programming languages	25
4.4	Evaluation	26
4.5	System overview	26
4.5.1	Reader module	27
4.5.2	Recommender module	29
4.5.3	Exporter module	29

5	Data	31
5.1	Description of the datasets	31
5.2	Number of interactions	34
5.3	Clusters	39
5.3.1	Compactness using k-means	39
5.3.2	Connectivity using Spectral Clustering	43
6	Parameter tuning	49
6.1	Training curves	49
6.1.1	katz-eig	50
6.1.2	link-analysis	51
6.2	Learning curves	52
6.3	Parameter space analysis	54
6.3.1	katz-eig	54
6.3.2	link-analysis	57
6.4	Optimized parameters	61
6.5	Algorithm comparison	62
6.5.1	katz-eig	62
6.5.2	link-analysis	65
6.5.3	Result	67
7	Discussion	71
7.1	Recommender systems	71
7.1.1	Future work	72
7.2	Datasets	73
7.3	Evaluation	74
7.4	Parameter tuning	75
7.4.1	Parameters of katz-eig	75
7.4.2	Parameters of link-analysis	76
7.4.3	Future work	77
8	Conclusions	79
A	Code	83
A.1	ESWC reader plugin	83
	Bibliography	85

1

Introduction

The introduction chapter presents the purpose and the goals of the thesis, what questions the thesis aims to answer, the limitations of the thesis and the contributions of this thesis. An outline of the thesis concludes the chapter.

1.1 Introduction

Being able to make choices, of any kind, has always been an important skill and perhaps it is more important now than ever before. It is hard to choose what products to buy, what music to listen to, what posts to read and what videos to watch as there are so many choices but a limited amount of time. In Youtube alone over 300 hours of video is uploaded every minute ¹.

This is why content providers and e-commerce are using recommendations, where items believed to appeal to the consumer are presented more prominently on the sites. Recommendations have become an important part of their business and companies such as Netflix are investing heavily into making their recommendations better ^{2 3}.

A common practice among e-commerce is to produce *related recommendations* where items are linked to related, similar, items. Another type is *personal recommendations* where items are recommended specifically for a single user given their interaction history.

There are simple algorithms to produce these recommendations, like recommending the most popular or the most watched movies. They are fast and easy to make but algorithms

¹Youtube Statistics, 2015. <http://www.youtube.com/yt/press/statistics.html>

²Netflix: Recommendations beyond 5 stars (Part 1), 2012. <http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html>

³Netflix: Recommendations beyond 5 stars (Part 2), 2012. <http://techblog.netflix.com/2012/06/netflix-recommendations-beyond-5-stars.html>

based on machine learning can produce more relevant recommendations. They work by learning from the data and building a model used to make predictions. The drawback is computational cost and complexity.

Explicit feedback recommender systems, which are concerned with ratings or other voluntary user feedback, have been researched extensively but *implicit feedback*, which passively collect information about the user, is not as extensively researched. [1, 2, 3]

This thesis examines the construction of a recommender system using implicit feedback and the evaluation of two different recommender algorithms, *link-analysis* and *katz-eig*. Both of the algorithms have their parameter space analysed and different optimization strategies are evaluated using several different datasets. The recommender system is built for Comordo Technologies as their core to later be built upon and extended.

1.2 Problem definition

The purpose of this thesis can be split in two larger parts. The first is to lay the foundation of Comordo Technologies' recommender system which could later be built upon and extended. At the end of this thesis the goal is to have a recommendation system which could load data supplied by Comordo's clients, produce recommendations and store them together with their recommendations in a database.

The second part is to analyze and create optimization strategies for *katz-eig* and *link-analysis* which optimize the algorithm's parameters for different datasets automatically. Parameter optimization should be done in a reasonable amount of time so the system can be commercially useful.

The recommendation algorithms depend on a couple of parameters which directly affects the quality of the recommendations made and the parameter values are different depending on the dataset the recommendations are being made for. Recommendation quality, or how good the recommendations are, is measured by the probability that a user interacts with the recommendation given by the system in the future where only recommendations to items not previously interacted with can be given. The goal of the optimization process is to maximize this probability for a specific dataset.

Core parts of the recommendation algorithms *katz-eig* and *link-analysis* existed before the thesis, but they were only runnable as Matlab scripts without any data handling and they lacked parameter tuning. There were also some optimization issues with the implementations. Focus is not on porting them to a different language or platform, which could improve them speed wise, but to adapt the existing code.

1.2.1 Guiding questions

These are some questions the thesis aims to answer.

- How can a recommender system be designed to allow for easily extendible input- and output handling?
- How can learning and recommendation using *link-analysis* and *katz-eig* be performed in practice, with regards to speed and recommendation quality?
 - How shall learning and optimization of their parameters be done?

To find an answer, an exploration of the function space of the parameters with regards to the evaluation criteria might be necessary.

1.3 Limitations

Although the goal is to handle real world data, the data considered in the thesis is of a limited size compared to the larger real world data. The implementations of the algorithms are not optimized enough to handle the larger data in a reasonable amount of time and under the memory limit of my machine⁴. It is possible to optimize the implementations by rewriting them or porting them to another language but it is outside the scope of this thesis.

This thesis focuses on *implicit feedback* systems with interaction history in *unweighted binary form* (Eq 2.1) which is also the focus for Comordo. *Explicit feedback* like ratings was not prioritized. Interactions in *weighted form* (Eq 2.5) might be interesting for Comordo, but it is not considered in this thesis. The *cold start problem* [4] is not considered in this thesis and no attempts are made to explain the recommendations.

Proprietary datasets used and code produced during the thesis will not be publicly released. See section 5.1 for a description of used datasets.

The purpose is to lay a foundation for Comordo's recommender system, but it does not include the remote API or the admin web interface (see section 4.2).

1.4 Contributions

A first version of Comordo's recommender system is built based around the recommender algorithms *katz-eig* and *link-analysis* with parameter optimization and flexible input- and output handling. The designed system can later be built upon and extended.

The parameter space over *F-measure* for *katz-eig* and *link-analysis* is analyzed for these datasets. An effective parameter optimization strategy for *katz-eig* is to fix β and to optimize K using a hill climbing algorithm. Similarly for *link-analysis* a good strategy is to fix η and optimize γ using an adaptive hill climbing algorithm.

⁴The test setup is described in section 6.

For sparse datasets *link-analysis* gives slightly better recommendations and for the other datasets *katz-eig* gives better recommendations. Speed wise *katz-eig* is superior. As the difference in recommendation quality for sparse datasets is so small *katz-eig* is the best general choice as the recommendations are better for the other datasets and it is generally much faster. The recommendations are better with datasets which have more interactions and worse for sparse datasets.

1.5 Outline of the report

This thesis consists of two parts. The first part concerns the system development part where a first version of Comordo's recommender system is built. The second part consists of an analysis of the parameter space and optimization strategies for the algorithms' parameters. The system development part is concentrated to chapter 4 and the parameter analysis to chapter 6.

Chapter 2 introduces the mathematical background for the thesis. The recommendation model and the learning process along with the recommendation algorithms *katz-eig* and *link-analysis* are presented.

Chapter 3 discusses work related to this thesis.

Chapter 4 covers the system development part of this thesis. Beginning with the given system development task and then presenting the constructed recommender system.

Chapter 5 presents the datasets used by this thesis. Contains an analysis of the datasets with respect to interactions and clusters.

Chapter 6 covers the parameter analysis and optimization. The chapter begins with an analysis of the algorithms and the parameter space and finishes with a comparison of different optimization techniques and a comparison between the algorithms.

Chapter 7 contains a discussion about the thesis and presents ideas for future work. Recommender systems in general and the one built are discussed. Then discussion about the datasets, the evaluation method and finally parameter tuning follow.

Chapter 8 concludes with the conclusions of this thesis.

Appendix A presents the available source code. Only an example reader plugin is available.

2

Background

This chapter introduces the mathematical theory behind recommendations and the recommendation model used by this thesis. The recommendation algorithms *katz-eig* and *link-analysis* are presented and a summary of machine learning follow which explains supervised learning and the evaluation metrics used. A section about optimization techniques finishes the chapter.

2.1 Recommendation theory

This section introduces the mathematical theory behind recommendations and it presents the two recommendation algorithms *katz-eig* and *link-analysis*.

This is the basic process of producing recommendations:

1. Given an interaction history $h_{u,i}$, $u \in Users$, $i \in Items$ and algorithm specific parameters the recommendation algorithm produce recommendations $p_{u,i}$.
2. The recommendations $p_{u,i}$, which are real values, are converted to binary recommendations $r_{u,i}$ by selecting the N largest $p_{u,i}$ as $r_{u,i} = 1$.

The process of parameter tuning used is as follows:

1. Split the interaction matrix A into a training set A_{train} , a validation set A_{val} and a test set A_{test} .
2. Evaluate different parameters by producing recommendations with A_{train} and evaluating them against A_{val} or A_{test} with respect to *F-measure*.
3. Select the best performing parameters with respect to *F-measure*.

2.1.1 Recommendation model

Given a set of users U , a set of items I and an interaction history $h_{u,i}$, $u \in U$, $i \in I$ given in *unweighted binary form*

$$h_{u,i} = \begin{cases} 1 & \text{if user } u \text{ has interacted with item } i \\ 0 & \text{otherwise} \end{cases} \quad (2.1)$$

the *recommender problem* is defined by producing a set of recommendations $r_{u,i}$

$$r_{u,i} = \begin{cases} 1 & \text{if item } i \text{ is recommended to user } u \\ 0 & \text{otherwise} \end{cases} \quad (2.2)$$

to maximize the probability that user u will want to interact with item i in the future, for all users and items. When $r_{u,i}$ is binary this is a *binary classification* problem. This definition is applicable for *implicit feedback* systems which passively track different sorts of user behaviour. For example link following, interaction time and purchase history.

As an additional constraint (Eq 2.3) no recommendations can be made for items already interacted with.

$$r_{u,i} = 0 \text{ whenever } h_{u,i} = 1 \quad \forall u, i \quad (2.3)$$

It is sometimes notationally convenient to treat the interaction history as a matrix. The whole interaction history $h_{u,i}$ will in matrix form be denoted by the interaction matrix $A = (h_{u,i})$, with each row representing each user and each column representing each item. The underlying structure forms a bipartite graph with one set representing the users and the other the items.

For example an interaction matrix

$$A = \begin{matrix} & \begin{matrix} i_1 & i_2 & i_3 & i_4 \end{matrix} \\ \begin{matrix} u_1 \\ u_2 \end{matrix} & \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{matrix}$$

with 2 users and 4 items corresponds to the interaction history: $h_{1,1} = 1$, $h_{1,3} = 1$, $h_{2,3} = 1$ and $h_{2,4} = 1$. The recommendation set $r_{u,i}$ will be represented by the recommendation matrix $R = (r_{u,i})$.

Implementation wise the matrices are often stored in a sparse format which only stores nonzero elements in memory. This can significantly speed up both computations and storage usage, depending on the sparsity of the matrix. The sparse format lends itself very well for interaction history in unweighted binary form (Eq 2.1) as the nonexistent interactions are modeled as zero elements in the matrix.

The recommender problem can be extended to the *Top-N recommender problem* by introducing constraints (Eq 2.4) (for a binary classifier) which states that only N recommendations can be presented for each user.

$$\sum_i r_{u,i} \leq N \quad \forall u \quad (2.4)$$

A variation of the recommender problem is when the interaction history is in *weighted form* (Eq 2.5), when the values increase with each interaction

$$h_{u,i} = \begin{cases} x & \text{user } u \text{ has interacted } x \text{ times with item } i \\ 0 & \text{otherwise} \end{cases} \quad (2.5)$$

for example $h_{u,i} = 2$ means that the user u has interacted with item i 2 times. It is possible to allow *implicit feedback* systems to log partial interactions, so $h_{u,i} = 0.7$ could mean that user u has watched 70% of the movie i , in the context of movie watching. [1]

The converse of *implicit feedback* is *explicit feedback* where the users give direct input regarding their preferences, for example with movie ratings or with likes and dislikes. Here the definition of the interaction history $h_{u,i}$ is the users' rating history (Eq 2.6).

$$h_{u,i} = \begin{cases} x & \text{the rating user } u \text{ gave item } i \\ \emptyset & \text{if the user } u \text{ did not rate item } i \end{cases} \quad (2.6)$$

With ratings $r_{u,i}$ changes to $r_{u,i} = \hat{x}$ where \hat{x} is the rating user u is predicted to give item i . This is also a *classification* problem, but the problem changes from assigning a binary value to wanting to predict a rating value.

To transform datasets with the more common explicit feedback style of ratings to an unweighted binary form a crude model (Eq 2.7) can be used.

$$h_{u,i} = \begin{cases} 1 & \text{user } u \text{ has rated item } i \\ 0 & \text{otherwise} \end{cases} \quad (2.7)$$

2.1.2 Recommendation prediction

The algorithms which produce binary classification recommendations produce predictions for each user-item pair, denoted $p_{u,i}$. Generally the higher the value of $p_{u,i}$ the more likely is it that user u will interact with item i . The predictions $p_{u,i}$ corresponds to the prediction matrix $P = (p_{u,i})$.

$p_{u,i}$ forms the bases for the recommendation set $r_{u,i}$. To produce Top- N recommendations take the N largest $p_{u_k,i} \forall i$ for each user u_k and set $r_{u_k,i} = 1$ for these N values. Set

$r_{u_k,i} = 0$ for the rest. It is possible to set $r_{u,i} = 0$ if $p_{u,i} \leq \epsilon$, for some ϵ , to accommodate for fewer than N recommendations.

In a classification context when the interaction history describes ratings the value corresponds to the predicted ratings user u would give i . The recommendations $r_{u,i}$ then becomes the closest discrete rating value of $p_{u,i}$. For example $p_{u,i} = 3.8$ means a user u is predicted to rate item i a 4, so $r_{u,i} = 4$, given discrete ratings between 1 and 5.

Some algorithms also output a confidence value $c_{u,i}$ which denotes how certain the predicted values are. This is relevant when predicting ratings, for example $p_{u,i} = 4.0$ may seem like a surely predicted 4 rating but a low value of $c_{u,i}$ means we might not want to recommend that item anyway.

2.1.3 The katz-eig algorithm

The *katz-eig* algorithm used is an adaptation [5] of a link prediction measure Katz [6]. Katz is defined as follows, if A is the interaction matrix and the measure is used to produce recommendation predictions

$$P = \sum_{t=1}^{\infty} \beta^t A^t = (I - \beta A)^{-1} - I \quad (2.8)$$

where I is the identity matrix. The intuition is that for each iteration t , one link in the interaction graph defined by user-item pairs is traversed and propagated to introduce transitive connections in the graph. The parameter $\beta \leq \|A\|_2$ represents the link dampening, links far away adds a smaller weight than links closer to the initial node.

The problem with this definition is computational complexity, computing the Katz measure takes $O(n^3)$ time which is not practical for large matrices. This is why the Singular Value Decomposition (SVD) is used.

A can be approximated by a rank k SVD so $A \approx U * S * V^T$. S is a $k \times k$ diagonal matrix with the elements representing the k largest singular values. Then the Katz measure can be approximated by

$$P = \sum_{t=1}^{\infty} \beta^t A^t \approx \sum_{t=1}^{\infty} \beta^t (U * S * V^T)^t \approx U \left(\sum_{t=1}^{\infty} \beta^t S^t \right) V^T \quad (2.9)$$

Exponentiation is moved from the large interaction matrix A to the small $k \times k$ diagonal matrix S which makes the iterative part of the algorithm very fast. Much of the information about the matrix is still contained in U and V . The complexity of the algorithm is now on calculating the SVD approximation.

Concretely the *katz-eig* algorithm follow these steps:

1. Construct U, S, V so $U * S * V^T$ forms a rank k SVD approximation of A . Let $S_0 = S$.
2. At each iteration $t = 1, \dots, t_{max}$ perform:
 - (a) $S_t = S_{t-1} + \beta^t * S_{t-1}^t$
 Repeat until convergence.
3. The prediction matrix is given by $P = U * S_{t_{max}} * V^T$.

Runtime example

This is a runtime example for *katz-eig* using a simple interaction matrix (Eq 2.10), which is the same matrix as in the example for *link-analysis* (see section 2.1.4).

$$A = \begin{matrix} & \begin{matrix} i_1 & i_2 & i_3 & i_4 \end{matrix} \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{matrix} \quad (2.10)$$

This example uses $K = 2$, $\beta = 0.1$ and is run for $t_{max} = 3$ iterations.

Firstly a rank 2 SVD approximation is created

$$U = \begin{pmatrix} -0.5592 & 0.4472 \\ -0.7805 & 0.0000 \\ -0.2796 & -0.8944 \end{pmatrix}, S = \begin{pmatrix} 2.1889 & 0 \\ 0 & 1.4142 \end{pmatrix}, V = \begin{pmatrix} -0.1277 & -0.6325 \\ -0.6120 & 0.3162 \\ -0.4843 & -0.6325 \\ -0.6120 & 0.3162 \end{pmatrix}$$

$$U * S * V^T = \begin{pmatrix} -0.2436 & 0.9491 & 0.1928 & 0.9491 \\ 0.2182 & 1.0455 & 0.8273 & 1.0455 \\ 0.8782 & -0.0254 & 1.0964 & -0.0254 \end{pmatrix} \approx A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Notable here is that some valid recommendations could be made directly from the approximation matrix. In fact for this example there would be no difference in recommendations if $t_{max} = 0$ and the approximation matrix was used directly.

Intuitively a matrix approximation blurs together similar users and items. In a rank k matrix approximation the value of k specifies the blurring degree, the higher k the more of the original matrix will be retained.

Initially $S_0 = S$. Then S_t is calculated iteratively

$$S_1 = \begin{pmatrix} 0.2189 & 0 \\ 0 & 0.1414 \end{pmatrix}, S_2 = \begin{pmatrix} 0.2668 & 0 \\ 0 & 0.1614 \end{pmatrix}, S_3 = \begin{pmatrix} 0.2773 & 0 \\ 0 & 0.1642 \end{pmatrix}$$

until convergence or as in our case $t_{max} = 3$. The recommendations are then given by

$$U * S_3 * V^T = \begin{matrix} & i_1 & i_2 & i_3 & i_4 \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} -0.0266 & 0.1181 & 0.0286 & 0.1181 \\ 0.0276 & 0.1324 & 0.1048 & 0.1324 \\ 0.1028 & 0.0010 & 0.1305 & 0.0010 \end{pmatrix} \end{matrix}$$

After removing the items users already have interacted with in A , the prediction matrix P becomes

$$P = \begin{matrix} & i_1 & i_2 & i_3 & i_4 \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} -0.0266 & 0 & \mathbf{0.0286} & 0 \\ \mathbf{0.0276} & 0 & 0 & 0 \\ 0 & \mathbf{0.0010} & 0 & \mathbf{0.0010} \end{pmatrix} \end{matrix}$$

Figure 2.1 is a visualization of P displaying the single most recommended item for each user.

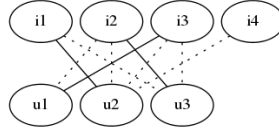


Figure 2.1: A graph representing the most recommended item for each user. The dotted lines represent the interaction history.

2.1.4 The link-analysis algorithm

The *link-analysis* algorithm is an adaption of a web page ranking algorithm HITS [7] to the recommendation domain [8, 9].

The original algorithm distinguishes between two types of web pages:

1. *Authoritative* pages which contain definite high quality information
2. *Hub* pages basically are lists of links to the authoritative pages

The authoritative score of a page is proportional to the hub scores linking to it. Similarly the hub score of a page is proportional to the authoritative scores of the pages it is linking to. These definitions are mutually reinforcing, good hub pages have links to many authoritative pages and good authoritative pages have links from many hubs.

Adaptation to the recommendation domain is achieved by introducing the *item representativeness* score IR and the *user representativeness* score UR . The difference between the recommendation domain and the web page ranking domain is that in the recommendation domain the aim is to produce personal recommendations where as in the web page domain generally popular pages are sought after.

The *item representativeness* score $IR(i, u)$ can be seen as a measure of the item i 's level of interest with respect to user u , or in other words i 's authority of u 's interests in i . This is an analogy to the authoritative page score. Intuitively if it is a high score then the item i can be recommended to user u .

The *user representativeness* score $UR(u, \hat{u})$ measures how well u as a hub for \hat{u} associates with items of interests to \hat{u} . This is analogous to the hub page score. Intuitively it is a measure for how similar the users u and \hat{u} are to each other.

A direct definition of the item and user representativeness scores is as follows where A is the interaction matrix:

$$IR = A^T * UR \quad (2.11)$$

$$UR = A * IR \quad (2.12)$$

There are two inherent problems with these definitions. The first is if a user has interactions with all items then that user will have the highest user representativeness UR for all users, even though such a user provides little information. The second problem is that IR and UR will converge to matrices with identical columns. This leads to item representativeness scores $IR(i, u)$ which are independent of the user u chosen and depend only on the item i . [8, 9]

To address these problems the user representativeness score is redefined [9] as

$$UR = B * IR + UR_0 \quad (2.13)$$

Where B is the normalization of the users A with respect to the total number of items the user has interacted with

$$B_{u,i} = \frac{A_{u,i}}{(\sum_i A_{u,i})^\gamma} \quad (2.14)$$

The effect of introducing B is that a user u with more item interactions than another user \hat{u} , to get a high user representativeness score $UR(u, \hat{u})$ the user u needs to have overlapping purchases with \hat{u} . The parameter γ controls the extent to which a user is penalized for making many purchases.

UR_0 is defined as a diagonal matrix with η on the diagonal. In other words $UR_0 = \eta * I_M$ where I_M is an $M \times M$ identity matrix and M is the number of users. It is included to maintain the high representativeness score for the target users themselves, which prevents IR and UR to converge to identical columns.

This also necessitates a normalization step of UR to keep the values on a consistent level, otherwise numerical problems could occur when the values keep growing.

In summary the *link-analysis* algorithm follow these steps:

1. Construct the interaction matrix A and the associating matrix B .
2. Set $UR_0 = \eta * I_M$.
3. At each iteration $t = 1, \dots, t_{max}$ perform:
 - (a) $IR_t = A^T * UR_{t-1}$
 - (b) $UR_t = B * IR_t$
 - (c) Normalize UR_t so each column adds up to 1
 - (d) $UR_t = UR_t + UR_0$

Repeat until convergence.

4. The predicted matrix is given by $P = IR^T$.

Runtime example

This is a runtime example for *link-analysis* using a simple interaction matrix (Eq 2.15), corresponding to the interaction graph figure 2.2.

$$A = \begin{matrix} & \begin{matrix} i_1 & i_2 & i_3 & i_4 \end{matrix} \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{matrix} \quad (2.15)$$

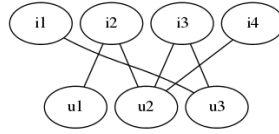


Figure 2.2: A graph representing the interaction history between each user and item, describing the interaction matrix in (Eq 2.15).

The following example uses $\gamma = 0.9$ and $\eta = 1$.

$$B = \begin{matrix} & \begin{matrix} i_1 & i_2 & i_3 & i_4 \end{matrix} \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 0 & 0.5359 & 0 & 0.5359 \\ 0 & 0.3720 & 0.3720 & 0.3720 \\ 0.5359 & 0 & 0.5359 & 0 \end{pmatrix} \end{matrix}, \quad UR_0 = \begin{matrix} & \begin{matrix} u_1 & u_2 & u_3 \end{matrix} \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{matrix}$$

During the iterations the rows of IR will be representing each item and each column will be representing each user, this is the reverse of the interaction matrix A . The example therefore presents the transpose of IR , IR^T .

$$IR_1^T = (A^T * UR_0)^T = \begin{matrix} & \begin{matrix} i_1 & i_2 & i_3 & i_4 \end{matrix} \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{matrix}$$

$$UR_1 = norm(B * IR_1) + UR_0 = \begin{matrix} & \begin{matrix} u_1 & u_2 & u_3 \end{matrix} \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 1.5902 & 0.4098 & 0 \\ 0.3935 & 1.4098 & 0.1967 \\ 0 & 0.2577 & 1.7423 \end{pmatrix} \end{matrix}$$

The first iteration does not alter the item representativeness matrix. In the user representativeness matrix links between users are made through one shared item. As seen in figure 2.3 a connection is made between u_1 and u_2 , through i_2 and a connection between u_2 and u_3 through i_3 .

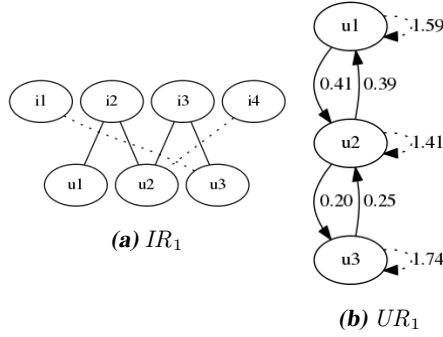


Figure 2.3: Visual representation of the first iteration. The full lines in UR_1 (b) represents new connections, which come from the new connections in IR_1 (a) also represented by full lines.

$$IR_2^T = (A^T * UR_1)^T = \begin{matrix} & i_1 & i_2 & i_3 & i_4 \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 0 & 2.0000 & 0.4098 & 2.0000 \\ 0.1967 & 1.8033 & 1.6065 & 1.8033 \\ 1.7423 & 0.2577 & 2.0000 & 0.2577 \end{pmatrix} \end{matrix}$$

$$UR_2 = norm(B * IR_2) + UR_0 = \begin{matrix} & u_1 & u_2 & u_3 \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 1.5354 & 0.4098 & 0.0548 \\ 0.3994 & 1.4008 & 0.1997 \\ 0.0858 & 0.2909 & 1.6233 \end{pmatrix} \end{matrix}$$

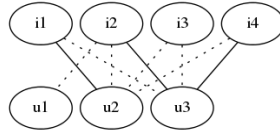


Figure 2.4: Visual representation of IR_2 . The full lines represents new connections.

New connections in IR_2 are made by using item connections from related users from UR_1 . In figure 2.4 new connections for u_3 are made to i_2 and i_4 because u_2 is now representing u_3 in figure 2.3b, and u_2 has connections with i_2 and i_4 .

$$IR_3^T = (A^T * UR_2)^T = \begin{matrix} & i_1 & i_2 & i_3 & i_4 \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 0.0548 & 1.9452 & 0.4646 & 1.9452 \\ 0.1997 & 1.8003 & 1.6006 & 1.8003 \\ 1.6233 & 0.3767 & 1.9142 & 0.3767 \end{pmatrix} \end{matrix}$$

$$UR_3 = \text{norm}(B * IR_3) + UR_0 = \begin{matrix} & u_1 & u_2 & u_3 \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 1.5234 & 0.4067 & 0.0699 \\ 0.3995 & 1.4007 & 0.1998 \\ 0.1226 & 0.3015 & 1.5759 \end{pmatrix} \end{matrix}$$

After transposing IR_3 and removing the items users already have interacted with in A , the prediction matrix P becomes

$$P = \begin{matrix} & i_1 & i_2 & i_3 & i_4 \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 0.0548 & 0 & \mathbf{0.4646} & 0 \\ \mathbf{0.1997} & 0 & 0 & 0 \\ 0 & \mathbf{0.3767} & 0 & \mathbf{0.3767} \end{pmatrix} \end{matrix}$$

Figure 2.5 is a visualization of P displaying the single most recommended item for each user.

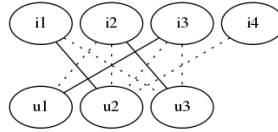


Figure 2.5: A graph representing the most recommended item for each user. The dotted lines represent the interaction history.

2.2 Machine learning

In this section a summary of supervised learning explaining how learning from the datasets is accomplished. A short summary of unsupervised learning, mainly focused on clustering, follows and metrics for evaluating recommendation quality is presented at the end of the section.

2.2.1 Supervised learning

The task of *supervised learning* is given a *training set* A_{train} with input-output pairs discover a function (or parameters for a function), the hypothesis, which approximates the input-output mapping. To measure the accuracy of the hypothesis match it against a *test set* A_{test} with input-output pairs distinct from the training set. [10]

The training set can be seen as the history available, what has happened before. The test and validation sets represents the future in a sense. Given the training set the task is to predict what happens “in the future”, stored in the test and validation sets.

In summary machine learning for supervised learning is done in a couple of steps:

1. **Preface** Split data set into training, test and validation sets.
2. **Training phase** Train the hypothesis, in our case select the algorithms’ parameters, using the training set.
3. **Model selection** Select model using the validation set. (Optional)
4. **Evaluation** Estimate the accuracy using the test set.
5. **Application** Apply the developed model to real world data and get results.

There can be multiple available models for the hypothesis, for example if the hypothesis is a polynomial function of the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0 \quad (2.16)$$

then the polynomial degree $n = 1, 2, 3, \dots$ represents different possible models for the hypothesis [10]. Other examples include the number of layers and the number of units in a neural network¹ or the rank of a low rank approximation².

The different models represents the complexity of the hypothesis. A more complex model can make a better fit to the training data but that introduces the problem of *overfitting* where the hypothesis fits the training data *too* well and it will not fit the test data. [10]

Model selection is the act of choosing a set of parameters, selecting a model, with the goal of optimizing the algorithm’s performance on an independent data set, a *validation set* A_{val} . The reason not to both choose the model and evaluate the model using the test

¹Machine Learning, Stanford. <https://class.coursera.org/ml-006>

²*katz-eig* models this way, see section 2.1.3

set is that then we will have overfit the test set as we both choose the best model and then evaluate with *the already best fit*. [10]

The recommended ratio to split the training, validation and test set differs but common recommendations include 60/20/20, 80/10/10, or 70/15/15 ³ depending on domain and the size of the available data set. It is important that the sets are pairwise disjoint.

If there is no need for a validation set, which can be the case if there are no models to choose from, common training/test set ratios include 70/30, 80/20 or 90/10 [1, 10] ⁴.

Another way to combat overfitting is with *regularization*. Regularization searches for a hypothesis which directly penalizes complexity. Regularization still needs to select the hyperparameter λ using model selection [10]. This will be explained further in section 2.3.

2.2.2 Unsupervised learning

In contrast with supervised learning, *unsupervised learning* doesn't have an expected output to learn from. Instead the task is to learn patterns in the input without any feedback. The most common unsupervised learning task is *clustering*: detecting potentially useful clusters, or groups, of input examples [10].

A common clustering technique is *k-means*, which clusters around k clusters [11]. Another technique is *spectral clustering* which is described in more detail in section 5.3 where it is used to find clusters the datasets.

2.2.3 Evaluation

A common technique to evaluate the accuracy, or the quality of recommendations, as sets is with *Precision*, *Recall* and *F-measure* ⁵ [2]. Evaluating as sets is done in the evaluation and model selection phase of supervised learning.

To evaluate between sets, let $r_{u,i}$ be the final recommendations in binary form (Eq 2.2) produced by the training set A_{train} . It's possible to evaluate Top- N recommendations by simply constraining the recommendation set $r_{u,i}$ (Eq 2.4). Let $e_{u,i}$ be the interaction history as described by the evaluation set. The evaluation set could either be the test set A_{test} or the validation set A_{val} , so $e_{u,i}$ should either represent A_{test} or A_{val} .

First define *true positives* TP as the sum of all correctly predicted positive samples.

$$TP = \sum_{u,i} r_{u,i} = 1 \wedge e_{u,i} = 1 \quad (2.17)$$

Conversely *false positives* FP is the sum of all falsely predicted positive samples.

³As recommended by Andrew Ng, Stanford. <https://class.coursera.org/ml-006>

⁴Andrew Ng also mentions these values

⁵The 2nd Linked Open Data-enabled Recommender Systems Challenge uses *F-measure*, 2015. <http://sisinflab.poliba.it/events/lod-recsys-challenge-2015/>

$$FP = \sum_{u,i} r_{u,i} = 1 \wedge e_{u,i} = 0 \quad (2.18)$$

And *false negatives* FN is the sum of all falsely predicted negative samples.

$$FN = \sum_{u,i} r_{u,i} = 0 \wedge e_{u,i} = 1 \quad (2.19)$$

Then *Precision* and *Recall* is defined as

$$Precision = \frac{TP}{TP + FP} \quad (2.20)$$

$$Recall = \frac{TP}{TP + FN} \quad (2.21)$$

Precision can be interpreted as how well the recommended items correspond to the users' actual preferences as described by the evaluation set and *Recall* signifies how well the users' preferences contained in the evaluation set fits with the recommendations.

In many ways precision and recall are competing measures, when optimizing for precision recall decreases and vice versa. As the number of recommendations N grow precision is expected to be lower and recall is expected to be higher. [1]

F-measure F1 is defined as the harmonic mean of precision and recall (Eq 2.22) as a combined measure of precision and recall.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (2.22)$$

Another evaluation method commonly used to evaluate classifications with ratings is the Root of Mean Square Error (RMSE). [2]

$$RMSE = \sqrt{\frac{\sum_{u,i}^n (r_{u,i} - e_{u,i})^2}{n}} \quad (2.23)$$

2.3 Optimization

Most supervised learning algorithms try to minimize a cost function during the learning phase. This function computes a value given some learned parameters and it can vary with different algorithms. The cost function does not make a comparison between two different sets but computes a metric from a single set.

A simple cost function (without regularization) could be defined as

$$\min_{r_{u,i}} \sum_{h_{u,i} \text{ is known}} (h_{u,i} - r_{u,i})^2 \quad (2.24)$$

A typical recommendation model associates each user u with a user-factors vector x_u and each item i with an item-factors vector y_i such that $r_{u,i} = x_u^T y_i$ [1]. In such a case a cost function could be defined as

$$\min_{x_u, y_i} \sum_{h_{u,i} \text{ is known}} (h_{u,i} - x_u^T y_i)^2 \quad (2.25)$$

where the the optimization objective is x_u and y_i . Usually stochastic gradient descent (SGD) is used to find the parameters [1]. With regularization a possible cost function could be

$$\min_{x_u, y_i} \sum_{h_{u,i} \text{ is known}} (h_{u,i} - x_u^T y_i)^2 + \lambda(\|x_u\|^2 + \|y_i\|^2) \quad (2.26)$$

where λ is the regularization hyperparameter found using *model selection*. This directly penalizes larger values of x_u and y_i which in this case corresponds to an increase in complexity.

Metrics such as *F-measure* can be used directly as optimization criteria if a suitable cost function is hard to find. It is also a common way of evaluating different models during model selection, the hyperparameter λ in equation (Eq 2.26) can be evaluated in this way⁶.

There are a couple of generic optimization techniques used for optimizing cost functions and selecting parameters via recommender quality metrics such as *F-measure*. In all cases the problem consists of minimizing or maximizing a target function. What follows are short descriptions of some common techniques:

Grid search

Grid search is a straightforward search technique which evaluates the function over a limited parameter space. This is a recommended approach for selecting the regularization

⁶Machine Learning, Stanford. <https://class.coursera.org/ml-006>

parameter λ ⁷.

Grid search is easily parallelized but it suffers from the curse of dimensionality, where it is particularly slow if used to optimize multiple parameters.

Random search

Grid search is exhaustive and possibly expensive, random search with a fixed limit of samples has been shown to be more effective in high-dimension spaces [12]. Random search is easily parallelized but lacks guidance.

Hill climbing

Hill climbing is a technique for finding a local optima from a given starting point. The neighbours of the current state are examined and the state is moved to the neighbour with a better function value until a local optima has been found. For continuous functions a variation called **adaptive hill climbing** exists which decrease the step size dynamically whenever a local optima is found to increase the precision. Other variations which incorporate random jumps exists, here collectively named **stochastic hill climbing**. [10]

Gradient based approaches

Variations of gradient based optimization techniques such as stochastic gradient descent can be used to optimize functions given that a gradient can be found. The search is similar to that of hill climbing, but is guided by the gradient and optimizes for a local optima. This is a fast and popular method for optimizing learning parameters. [1]

Simulated annealing

Simulated annealing is a probabilistic heuristic optimization technique used for finding global optima in a limited search space. It works by randomly jumping to neighbouring points with decreasing probability until it converges on a local optima. However it is more likely to find a better local optima than a gradient based approach. [10]

Bayesian optimization

Bayesian optimization develops a statistical model over the function space and evaluates the function sparsely which balances exploration and exploitation. With *Gaussian process priors*, a form of statistical modeling of a function, Bayesian optimization has been shown to give better results with fewer evaluations than grid search. [13]

⁷Suggested by Andrew Ng in his lectures on Machine Learning. <https://class.coursera.org/ml-006>

3

Related work

A lot of research has been put into recommender systems [2, 3, 14, 15, 16]. Most articles are concerned with improving accuracy of recommender system results, such as minimizing *Root of Mean Square Error* (RMSE). This was the case for the popular Netflix Prize [15] which was concerned with recommending movies given user ratings for other movies. *Explicit feedback* recommender systems continue to be a well researched area [2, 3, 14, 16]. *Implicit feedback* systems, which is the focus of this thesis, have grown in popularity and are being actively researched but is still less researched than *explicit feedback* [1, 2, 3].

According to [17] the *Top-N Recommendation problem* is the real problem of many on-line recommender systems and it is common to seek improvements for recommendation quality, using *Precision*, *Recall* or *F-measure* [2, 3, 18]. This is also the focus of this thesis.

The 2nd Linked Open Data-enabled Recommender Systems Challenge ¹ is another competition which focuses on improving recommendation quality using *F-measure* for the Top-N Recommendation problem as well as additional objectives such as *diversity* [2]. The recommender system challenge uses *explicit feedback* in the form of likes, but the data format is compatible with the model (Eq 2.1) this thesis uses. They use item meta-data such as genres, albums and actors which is not applicable to the general *implicit feedback* system this thesis is focused on.

Together with the research many versions of different recommender systems have been implemented, with recommender systems becoming more and more popular [2, 16]. One of the most popular types are hybrid recommender systems which combine different types of data and algorithms [2, 17]. This was the winning approach for the Netflix Prize which

¹2nd Linked Open Data-enabled Recommender Systems Challenge, 2015. <http://sisinflab.poliba.it/events/lod-recsys-challenge-2015/>

combined 107 different algorithms in different ways to produce the final recommendations².

Optimization strategies for parameter tuning differ depending on the algorithm. Alternating least squares (ALS) is a popular recommendation algorithm used both in *explicit feedback* and *implicit feedback* systems [1, 3]. Stochastic gradient descent (SGD) is a popular optimization strategy for ALS [1, 3] but there is also a custom optimization strategy purely for ALS [3]. Another popular approach is bayesian personalized ranking which can also be optimized with SGD [19].

No literature concerning parameter optimization could be found for either *link-analysis* nor *katz-eig*. Grid search seems to be the recommended approach for optimizing hyperparameters³. For *implicit feedback* systems the optimization of common cost functions⁴ is computationally expensive [3].

The *link-analysis* algorithm compared favorably in recommendation quality by [9]. But without any analysis of the algorithms' parameters. The parameter values are simply stated but not commented on any further. No literature with further comments on the parameters could be found. Relatively poor runtime performance is noted [8] but no actual comparisons are found.

Similarly *katz-eig* had some positive recommendation quality results [5] but without parameter analysis and no mention of the algorithm's speed. No literature for parameter tuning could be found.

²Netflix: Recommendations beyond 5 stars (Part 1), 2012. <http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html>

³Recommended by Andrew Ng for the course Machine Learning, Stanford. <https://class.coursera.org/ml-006>

⁴Similar to the definition in (Eq 2.24).

4

The Comordo recommender system

The thesis can be split in two major parts. The first part is a system development part where a first version of Comordo's recommender system is built, the "glue" around the recommender algorithms. The second part with development of a learning framework for the algorithms and an analysis of the algorithms' parameters. This chapter describes the system development part.

First is some background information about Comordo Technologies and the task given by Comordo for the construction of the recommender system which is the main purpose of the thesis from Comordo's point of view. A system sketch provided by Comordo and a use case of their product is included. Then follows the development methodology used during this thesis and how evaluation of the recommendations is done. The final section presents the developed recommender system and its modules.

4.1 Comordo

Comordo Technologies is a startup in recommendation systems driven inside the bounds of LiU's incubator LEAD in Linköping and will in the future offer a cloud service for e-commerce. At the start of this thesis the company stood to build a first version of their recommendation system, which is the purpose of this thesis.

Comordo focuses on generating personal recommendations using *implicit feedback* aimed at e-commerce using purchase history for users as their main focus. The end product aims to be a remote API where e-commerce clients queries for recommendations for their users.

4.2 System development task

The system development task for this thesis is to complete the backend of Comordo's system. This includes the reader, input, output and parameter modules, the storage of purchase history and parameters and modules for parameter tuning. The other databases were provided, but some level of adaptation was needed. The recommender algorithms *katz-eig* and *link-analysis* were given and again some adaptation was needed. The admin interface and the remote API are outside the scope of this thesis.

Figure 4.1 is the system sketch of Comordo's recommender system, as planned for at the start of this thesis.

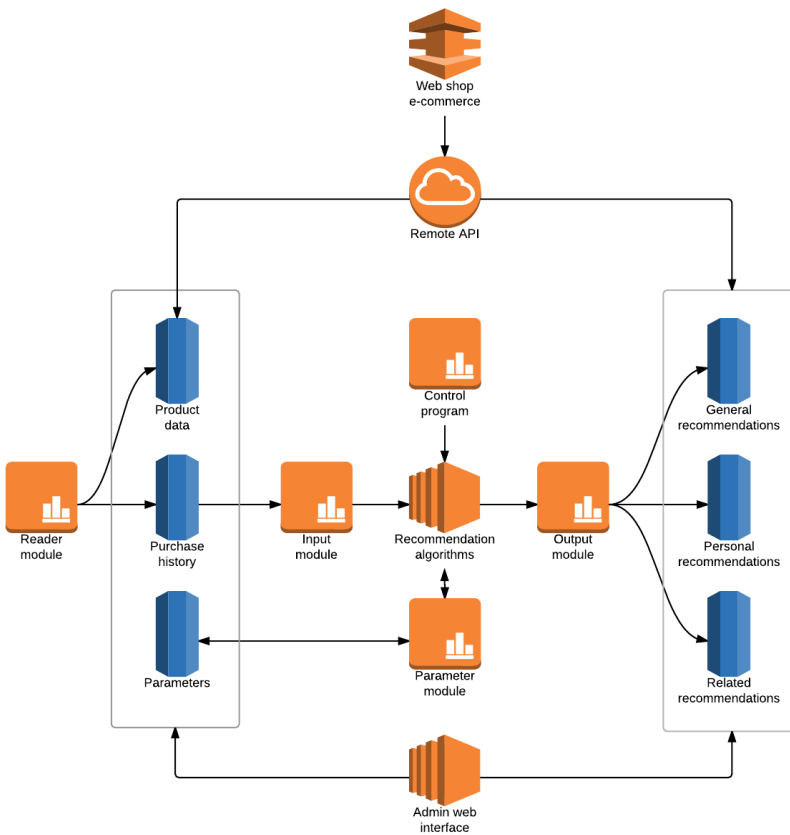


Figure 4.1: Comordo's system sketch

Reader module is responsible for reading data files provided by Comordo's clients.

Input module provides the algorithms with transformed data.

Output module populates the database with recommendations.

Control program handles learning and optimization of the algorithms.

Parameter module stores and adjusts parameters the algorithms use.

Remote API is a REST based API, the endpoint for Comordo's clients.

Admin web interface is a user friendly way for e-commerce clients to customize system settings and view recommendations.

4.2.1 Use case

This is a high level use case for how Comordo's recommender system will be used via the remote API and how recommendations will be produced for Comordo's clients.

1. Purchase history and product data is provided by e-commerce clients and consumed by the recommender system.
2. Load algorithms with purchase history and produce recommendations.
3. Repopulate recommendation database with new recommendations.
4. Final customers visit the e-commerce website and are given recommendations delivered to the website via Comordo's remote API.

4.3 Development methodology

The software is developed using agile inspired methods. Iterative development is used to produce a simple prototype and then iteratively improved upon with more features. The priority early on is to produce a working chain from reading data to storing recommendations in the database.

Small incremental goals are used, for example to complete a reader plugin for a specific dataset. Automatic tests and unit tests are used but not in the test driven development way, the requirement for the tests being made before the functionality was relaxed and not required.

4.3.1 Programming languages

The existing algorithms exists in a prototype form in Matlab. The thesis continued to use the algorithms written in Matlab for easy prototyping and modifications. Python was used as glue and to implement all modules, see section 4.5.

Usage of other languages or platforms, such as Julia, C, C++, or Python with NumPy or SciPy could give performance improvements, but it is outside the scope of this thesis. It was valuable to continue with a platform familiar to Comordo as they are in the startup phase with a focus on prototyping and performance enhancements can come later.

4.4 Evaluation

Recommendation quality is evaluated using *Precision*, *Recall* and *F-measure* with top-10 recommendations, as described in section 2.2.3. Focus is on *F-measure* as a combined measure of *Precision* and *Recall*.

The following steps describes the steps taken to produce evaluations given the training, validation and test sets A_{train} , A_{val} and A_{test} :

1. Produce recommendation predictions matrix P from A_{train} with the chosen algorithm.
2. Transform P to the recommendation matrix R using the top-10 most predicted items for each user.
3. Evaluate *F-measure* with $e_{u,i}$ representing A_{val} or A_{test} , depending on which set to evaluate against.

The validation set A_{val} is used to evaluate the choice of k as the rank- k SVD approximation in *katz-eig*. All other evaluations are done against the test set A_{test} .

4.5 System overview

Some changes are made to Comordo's original system design, as given in section 4.2. The final system is shown in figure 4.2.

The logic of the recommender system is built of two major parts: the *reader module* and the *recommender module*. Several modules from the original sketch has become submodules inside the recommender module. This is a logical grouping as the reader module and the recommender module are both implemented as separate scripts and the submodules represents a higher level description of the implemented functionality.

The *exporter module* is an utility module which generates recommendations from the database into another output format and serves statistics and as a developer debugging tool. The remote API and the admin web interface are included in the system sketch, but they are not implemented by this thesis.

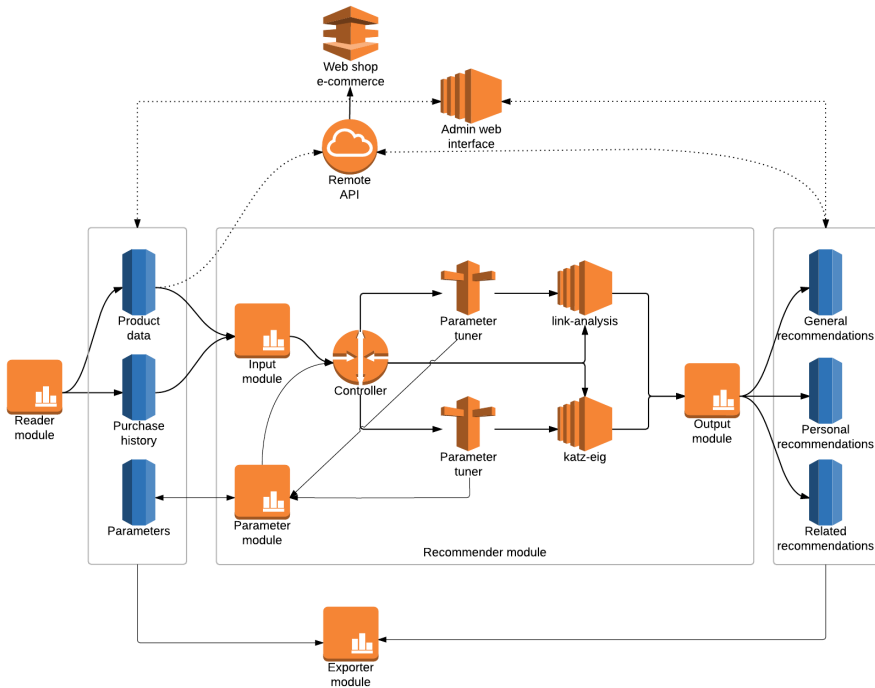


Figure 4.2: Overview of the recommender system. Dotted lines represents interactions not implemented by the thesis and the thick lines depicts the flow of generating recommendations.

4.5.1 Reader module

The reader module takes data files, with client specific formatting, and stores the data in the databases. The data contains user interaction history of some sort, possibly as a list of user-item pairs, but it can also contain additional user and item information all in a single file or in several.

To allow for flexibility the reader module uses a plugin system which can be selected at runtime. This is accomplished using python's dynamic module loading capabilities.

Firstly the reader module will get a list of available plugins found in

```
lib/reader_plugins
```

The plugin class shall have a single uppercase letter and the rest lowercase and reside in a file with all lowercase. For example a plugin which handles *eswc* data could have the class "Eswc" inside a file "eswc.py" in the plugin directory.

Secondly the appropriate plugin will be selected via command line arguments and the plugin class will be handed control. The class should have two methods: "add_arguments"

which parses extra command line arguments and “load” which shall return a user hash and a product hash. Appendix A.1 describes a full example plugin which handles *eswc* data.

With the selected data the reader module can then generate Matlab data file output in the form of a “.mat” file, upload the data to the database or simply print some statistics. When generating a “.mat” file different ratios of training, validation and test sets can be set. The purpose of this option is to generate datasets used during prototyping and evaluation.

The reader module can remove items and users from the dataset by introducing a couple of constraints:

1. limit the maximum number of users in the dataset
2. limit the maximum number of items in the dataset
3. remove users with too few item interactions in their history
4. remove items which too few users has interacted with

The reason to limit the size of the dataset is due to the high computational complexity and the bad performance on large datasets. The removal of items or users with too few interactions is because of the difficulty of generating recommendations for items or users with no history. This is known as the *cold start problem* and it is a known difficulty in recommendation systems [4] outside the scope of this thesis.

The reader module tries to conform to the constraints with these steps:

1. Remove users with too few items in history, if required to
2. Remove items which too few users has interacted with, if required to
3. Limit the number of items, if required
 - (a) Randomly select the items to keep
4. Limit the number of users, if required
 - (a) Randomly select the users to keep
5. Perform step 1 again
6. Perform step 2 again

This will not produce a perfect solution and some constraints may not be fulfilled. If we for example want to constrain both the minimum number of item interactions each user has and the minimum number of user interaction each item has, we might fail to find a solution as the removal of some items may cause some users to have fewer than the constrained number of item interactions.

The alternative is to introduce a constraint solver or iteratively perform step 1 and 2 until convergence, but that’s a slow solution to a problem with inherently soft constraints. It is not very important if *all* constraints hold, it is just an attempt to limit the size of the dataset. Therefore a faster but less correct heuristic is chosen.

4.5.2 Recommender module

The recommender module is the core of the recommender system. It is responsible for populating the databases with new recommendations and for optimizing the algorithms' parameters to new datasets.

Below follows a short description of the different submodules and their function.

Input module reads the interaction history from the database.

Controller selects which algorithm to use and if the purpose is to optimize the parameters or to generate recommendations.

Parameter tuner is responsible to optimize and fit the algorithms' to a new dataset.

link-analysis, katz-eig are the available recommender algorithms.

Output module populates the database with new recommendations.

When learning parameters the recommender module stores the found optimal parameters in the database. Then when generating personal recommendations the stored parameters can be used.

As an additional feature apart from generating personal recommendations, the recommender module can populate the database with general recommendations, which recommends the most popular items, and related recommendations which creates recommendations on an item level.

4.5.3 Exporter module

The exporter module's main function is to generate recommendation output in a file format. This serves as both a workaround for the lack of a working remote API and as an extra feature as Comordo's e-commerce clients might request the recommendations in a file format. It acts as an easy way of creating a formatted database dump.

The secondary function is to serve statistics and act as a developer debugging tool. Examination of the dataset and the generated recommendations can be made. It can be used to examine a user, the history and the recommendations generated.

5

Data

This chapter lists and describes the datasets used by the thesis, their contents and, if possible, where to find them.

Then the data is analyzed in two ways. Firstly the number of interactions is examined, both with respect to users and to items. All datasets are found to be top heavy, with *alphaS* less so, with few very popular items encompassing most of the user base. Secondly clusters in the datasets is searched for with respect to compactness, or user similarity, using *k-means* and on connectivity using *spectral clustering*. Connected clusters are identified in all datasets.

5.1 Description of the datasets

These are the datasets used by the thesis, a summary of the available datasets and their size is given by table 5.1. Some of the datasets (*alpha*, *alpha2*, *alphaS*, *romeo*) are given by some of Comordo's clients and they do not want the data to be publicly available. Instead a high level description of the datasets are given.

All of the datasets will be in unweighted binary form (Eq 2.1). Some of the datasets (*alpha*, *alpha2*, *alphaS*) support weighted form (Eq 2.5) but the other datasets does not, so they are transformed into unweighted binary form. Another given format is ratings, which *movielens1m* use. Generating recommendations with explicit feedback, such as ratings, is well researched but fundamentally different from implicit feedback systems. The focus of this thesis is on implicit feedback systems which is why ratings are not considered in their raw form, they are converted to unweighted binary form.

During supervised learning the datasets will be divided into training, validation and test sets with a ratio of 70%, 15% and 15% respectively. The split is done by randomly

distributing all items in the interaction history and distributing amongst the sets. In a matrix representation it can be thought as randomly assigning each non-zero value from the interaction matrix A to either A_{train} , A_{val} or A_{test} while keeping all other elements as zero.

When a validation set is not necessary, it will be ignored and only the training and test sets will be used. This is done for simplicity and to reduce the number of different datasets needed to keep track of. As mentioned in section 2.2.1 there are different ratios commonly used to split datasets. There is no ratio which is always the best, they depend on the amount of data available, the modeled domain and the algorithms chosen. A split of 70/15/15 is chosen early for simplicity reasons.

dataset	users	items	elements	sparsity
<i>alpha</i>	100002	219767	904201	0.0041%
<i>alpha2</i>	75007	345674	1945115	0.0075%
<i>alphaS</i>	16444	5000	26035	0.0316%
<i>eswc2015books</i>	1398	2609	11600	0.32%
<i>eswc2015movies</i>	32169	5389	638268	0.37%
<i>eswc2015music</i>	52072	6372	1093851	0.33%
<i>movielens1m</i>	6040	3706	1000209	4.5%
<i>romeo</i>	8321	722	205534	3.4%

Table 5.1: A summary of the used datasets

What follows is a description of each available dataset and where they can be found, if applicable.

alpha, alpha2, alphaS

Anonymous datasets representing purchase history provided by an e-commerce client. The dataset is given in a weighted form (Eq 2.5) but is converted to un-weighted binary form (Eq 2.1).

alpha is a randomly sampled dataset. It contains 100002 users, 219767 items with 904201 interactions.

alpha2 is another randomly sampled dataset, independently sampled from *alpha*, filtered to only contain users with ≥ 2 purchases. It contains 75007 users, 345674 items with 1945115 interactions.

alphaS is a subset of *alpha2*. It contains 16444 users, 5000 items with 26035 interactions. This is often used as *alpha* and *alpha2* are very large and the runtime is very long.

eswc2015movies, eswc2015music, eswc2015books

These are the datasets used in the 2nd Linked Open Data-enabled Recommender Systems Challenge ¹. The data have been collected from Facebook profiles about

¹2nd Linked Open Data-enabled Recommender Systems Challenge, 2015. <http://sisinflab.poliba.it/events/lod-recsys-challenge-2015/>

personal preferences, "likes", for movies, books and music. ².

The datasets were originally split into training sets and evaluation sets. The evaluation sets does not contain any user-product mappings and for evaluation purposes this thesis will only concern itself with the training set part of the datasets.

eswc2015books contains 1398 users with 11600 likes for 2609 items. The dataset contains likes for books, characters, genres and writers.

eswc2015movies contains 32159 users with 638268 likes for 6389 items. The dataset contains likes for movies, actors, directors, characters and genres.

eswc2015music contains 52072 users with 1093851 likes for 6372 items. The dataset contains likes for albums, artists, bands, compositions and genres.

For the purpose of this thesis, the different item types are treated as a single type. For example no care is taken to cross-reference liked genres with movies in that genre. The only thing considered is the unweighted binary user-item interaction history.

movielens1m

The MovieLens 1M dataset ³ is a collection of ratings (1-5) taken from the MovieLens website ⁴.

Ratings are transformed to unweighted binary form using (Eq 2.7).

This is by no means a perfect transformation as a rating of 1 means the user has consumed an item but didn't enjoy it, while our model only concerns itself with interactions. Noise is introduced into the dataset and the recommendations loose relevance with respect to the original unmodified dataset. It is still possible to evaluate recommendation using *F-measure* with respect to the new dataset in unweighted binary form, but no relevant conclusions can be made for the users themselves.

The dataset contains 6040 users with 1000209 ratings for 3706 movies.

romeo

An anonymous dataset representing purchase history provided by an e-commerce client. The dataset is in unweighted binary form.

The dataset contains 8321 users, 722 items and 205534 interactions.

Some of the datasets are very large and later in the thesis all datasets are not used as the runtime is so long. For example not all datasets are handled in the parameter analysis in section 6.3. Specifically the datasets *alpha*, *alpha2* and *eswc2015music* are often excluded and *eswc2015movies* is also excluded sometimes.

²DataSet | 2nd Linked Open Data-enabled Recommender Systems Challenge, 2015. <http://sisinflab.poliba.it/events/lod-recsys-challenge-2015/dataset/>

³GroupLens: MovieLens dataset, 2015. <http://grouplens.org/datasets/movielens/>

⁴MovieLens homepage. <https://movielens.org/>

5.2 Number of interactions

What follows is some plots describing the number of interactions each user has and the number of interaction each item has in the datasets. It's useful for identifying outliers and possibly for identifying defining features of a dataset.

The plot on the left describes how many users have a fixed number of item interactions and conversely the plot on the right describes how many items have a set number of user interactions. The histograms are also represented in logarithmic scale.

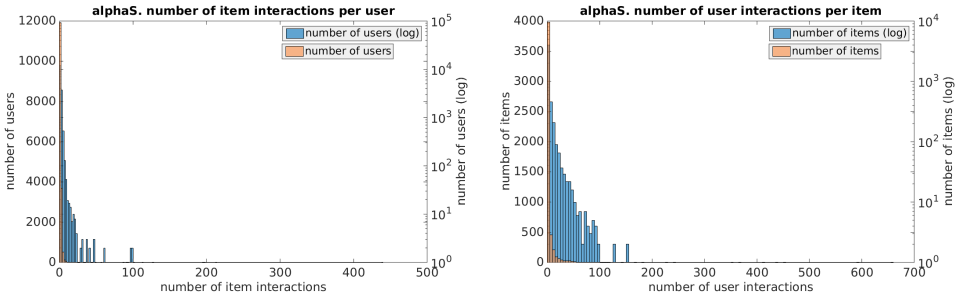
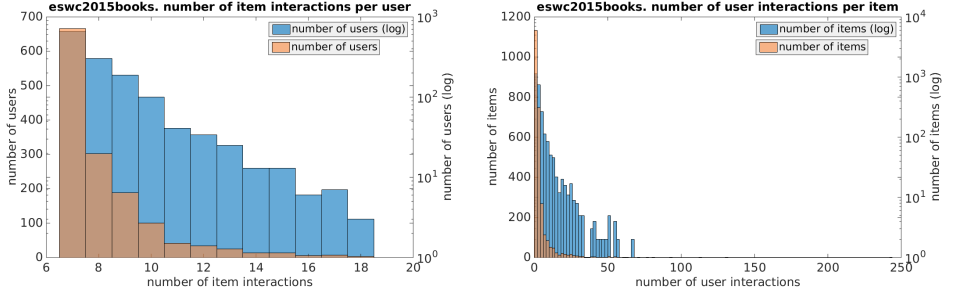
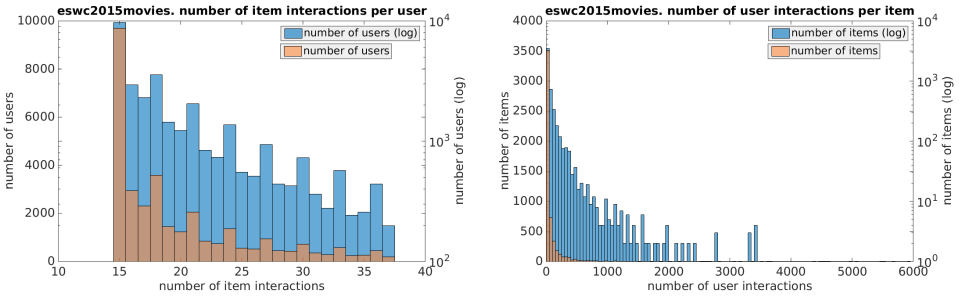
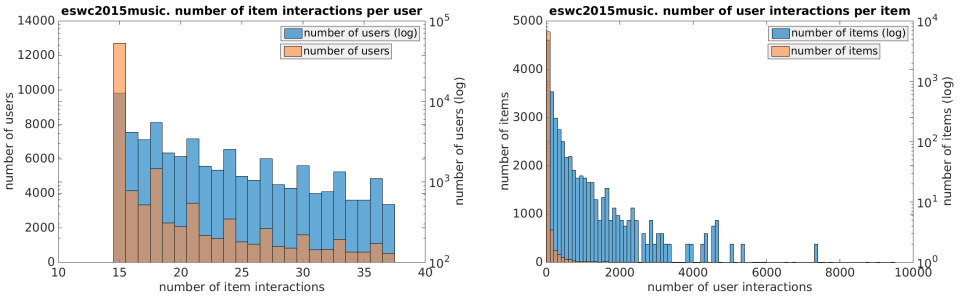


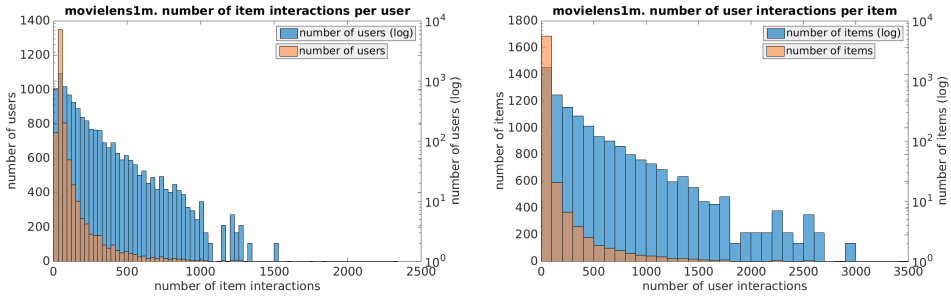
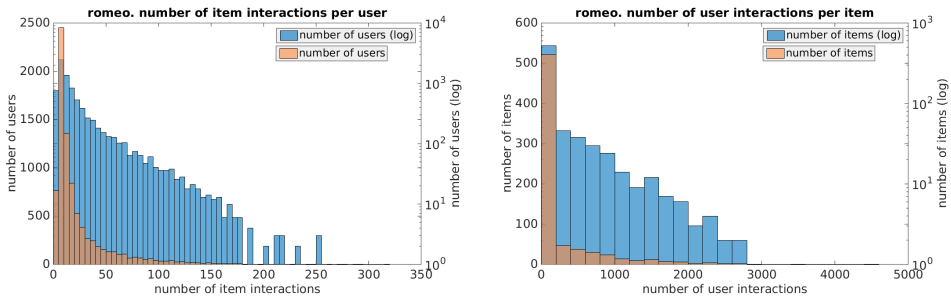
Figure 5.1: *alphaS*

In *alphaS* each user and each item has interactions with a small fraction of the available items and users. There are many users with very few interactions and also many items which few users has interacted with. There are no users or items without any interactions, but there are 11923 out of 16444 users and 2588 items out of 5000 with with only one interaction. This can be compared to the 26035 total interactions in the dataset. There are some users with more than 400 interactions and some items which have interacted with more 600 users.

Figure 5.2: *eswc2015books*Figure 5.3: *eswc2015movies*Figure 5.4: *eswc2015music*

All *eswc* datasets have similar distributions with more concentrated interactions. There is a lower limit for the number of interactions each user has, this is probably a constraint used when the datasets were made. There are also no extreme user outliers with many more interactions than the norm.

The item interactions are more spread, with many items having interacted with relatively few user but some items having a lot of interactions. *eswc2015books* have 1134 out of 2609 items with only one user interaction. *eswc2015movies* and *eswc2015music* in comparison have 2 out of 5389 items and 1 out of 6372 items with one user interaction.

Figure 5.5: *movielens1m*Figure 5.6: *romeo*

Both *movielens1m* and *romeo* have a more normalized look to them, especially with the number of user interactions per item compared to the other datasets. There are still outliers with many more interactions however. There are no users with less than 2 item interactions and there are no items without a user interaction. 114 out of 3706 items and 17 of 722 items have 1 user interaction in *movielens1m* and *romeo* respectively.

In general two distinct types of users can be identified. The first is a user with only a couple of item interactions, this appears to be the most common type of user. It could possibly be users who try out a service but for some reason they do not continue or they are new users who just recently started using the service. The other user type is the one with a lot of item interactions, way more than the norm, and they are quite rare⁵. They do not exist in the *eswc* datasets.

A similar classification can be made for items. The vast majority of items has only a couple of user interactions. Perhaps these are new items few users have found out about or niche items not interesting to most users. A large fraction of the items in *alphaS* and *eswc2015books* have only one interaction (51% and 43%). Then there are items with a lot more user interactions than what is common.

⁵Parallels can be drawn to what is known as big spenders or “whales” in the social-gaming community. They make up a tiny group of the community but they drive most of the revenue for the game publishers. For a more in depth discussion see

VentureBeat: What it means to be a “whale” — and why social gamers are just gamers, 2013.

<http://venturebeat.com/2013/03/14/whales-and-why-social-gamers-are-just-gamers/>

The following plots display the number of the most popular items and the number of users they collectively interact with. This is useful for investigating how top heavy the datasets are. The dashed lines represents the number of items required to include 95% of all users, a summary of the required number of items can be found in table 5.2.

dataset	items needed	items total	item ratio
<i>alphaS</i>	3058	5000	61%
<i>eswc2015books</i>	120	2609	4.6%
<i>eswc2015movies</i>	55	5389	1.0%
<i>eswc2015music</i>	78	6372	1.2%
<i>movielens1m</i>	13	3706	0.35%
<i>romeo</i>	21	722	2.9%

Table 5.2: This table describes how many of the most used items are necessary to include in a set so 95% of all users have interacted with the set.

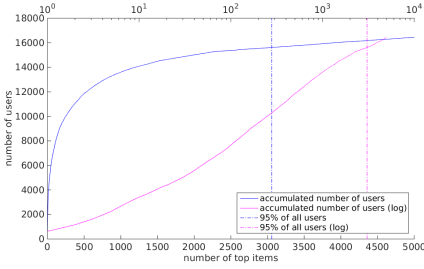


Figure 5.7: *alphaS*. 3058 of 5000 (61%) of the items are necessary to include 95% of all users.

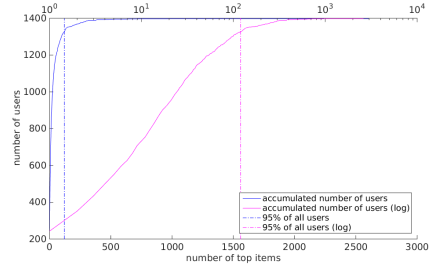


Figure 5.8: *eswc2015books*. 120 of 2609 (4.6%) of the items are necessary to include 95% of all users.

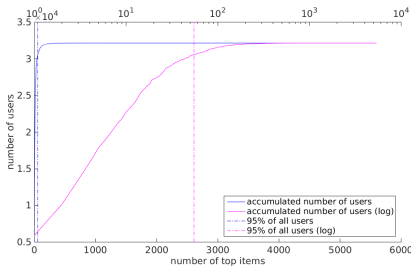


Figure 5.9: *eswc2015movies*. 55 of 5389 (1.0%) of the items are necessary to include 95% of all users.

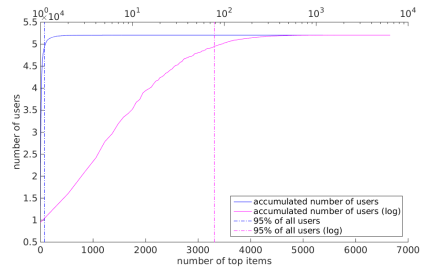


Figure 5.10: *eswc2015music*. 78 of 6372 (1.2%) of the items are necessary to include 95% of all users.

Of the different datasets, *alphaS* is a clear outlier. It is nowhere near as top heavy as the other datasets are, requiring over 60% of all items to reach 95% of the users. This can

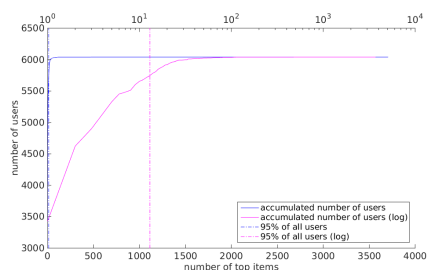


Figure 5.11: *movielens1m*. 13 of 3706 (0.35%) of the items are necessary to include 95% of all users.

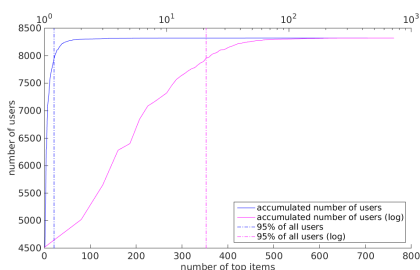


Figure 5.12: *romeo*. 21 of 722 (2.9%) of the items are necessary to include 95% of all users.

in part be explained by the large number of users with only one interaction, 11923 out of 16444 users or in other words 72% of all users have only one item interaction.

In contrast *eswc2015books* require 4.6% of the items and *romeo* require 2.9% of the items, which means few of the popular items are required to include most of the users. The other datasets are even more top heavy with *eswc2015movies* and *eswc2015music* only require 1.0% and 1.2% of the items. For *movielens1m* only 0.35%, namely 13, of the items are needed. In other words this means that 95% of all users in the dataset has seen at least one movie from the 13 most watched movies.

This phenomena where very few of the most popular items command the attention of most of the user base is also seen in mobile app stores where 1.6% of app developers make more than the other 98.4% combined ⁶.

⁶readwrite: Among Mobile App Developers, The Middle Class Has Disappeared, 2014.
<http://readwrite.com/2014/07/22/app-developers-middle-class-opportunities>

5.3 Clusters

Clustering with regards to compactness is examined using *k-means* and with regards to connectivity using *spectral clustering*. Compactness refers the closeness in space of the nodes in each cluster and connectivity refers to how connected the nodes in each cluster are with each other. In the recommendation domain compactness refers to how similar the users' full interaction history is to each other and connectivity instead examines similarity over user-item links.

As an example for two users who are close with respect to connectivity but not to compactness is when one user has a small subset of the other user's interactions.

5.3.1 Compactness using k-means

This is the clustering process used to cluster on a user level with *k-means*. The goal is to reorder the users so similar users are situated next to each other in the interaction matrix A .

1. Approximate the interaction matrix A by a rank k SVD approximation, $U * S * V'$ where S is a $k \times k$ matrix representing the k largest singular values.
2. Locate k clusters with *k-means*, operating on $U * S$.
3. Reorder the rows, representing the users, in A using the clustering information.

A k rank approximation of A is used for two reasons. The first is to remove noise and to introduce similarities for items. U spans the "column space" which is the reason for operating on $U * S$ as we want to cluster on a user level. The second reason is for speed reasons.

k-means clusters, or classifies, 2D-points into k clusters. Using the clustering information for the users, each row in the interaction matrix A are reordered so all users in the same cluster are next to each other.

For *k-means* the hard part is finding a good k value. This experiment is more concerned about finding information about any clusters, not about finding the optimal amount, so the number of clusters $k = 10$ is fixed.

The plots display a clustered interaction matrix, reordered user wise. If there are visible clusters the expected thing to see is horizontal bands of similar user patterns. Vertical bars represents items, or a set of items, with many interactions. Some of the plots will be sparse and some will appear to be very dense. This is mostly due to resolution issues as when the dataset grow, even though the sparsity might be lower, the number of data points grow but the size of each data point in the plots are the same.

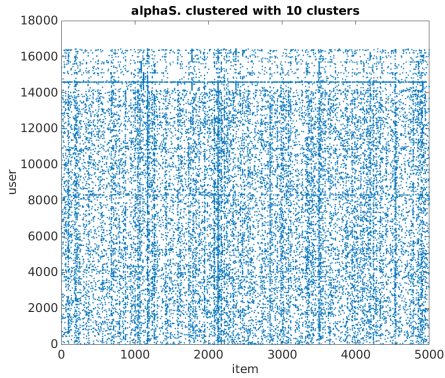


Figure 5.13: *alphaS*

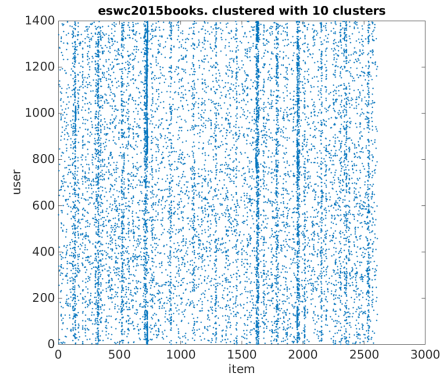


Figure 5.14: *eswc2015books*

In *alphaS* there are some clusters with relatively few item interactions, but the other clusters aren't very prominent. Both *alpha* and *alpha2* are so large the resolution isn't enough to capture any individual data points so their plots don't show anything of value so they are not included here. *eswc2015books* doesn't seem to have any major clusters, the dataset doesn't appear to have any structure except for a few streaks of very popular items.

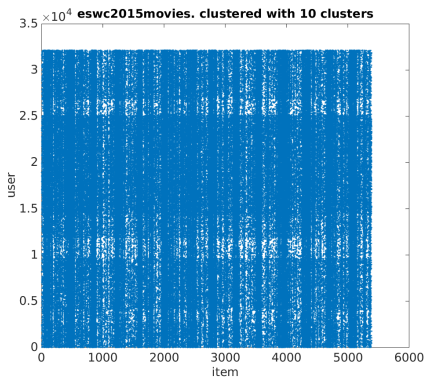


Figure 5.15: *eswc2015movies*

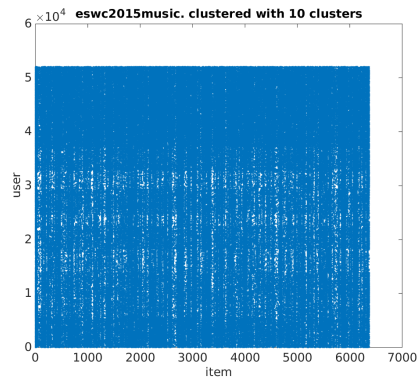


Figure 5.16: *eswc2015music*

eswc2015movies and *eswc2015music* in contrast display more prominent clusters. There are clusters who concentrate more on a subset of items, and there are clusters with higher interaction count. Similarly *movielens1m* and *romeo* appear to have some clusters.

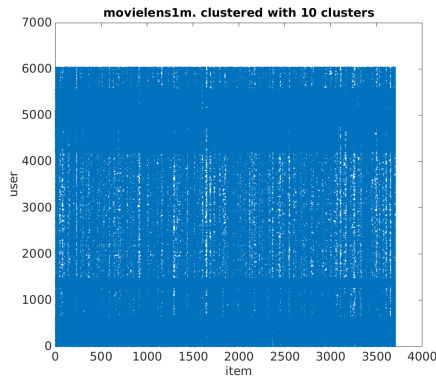


Figure 5.17: *movielens1m*

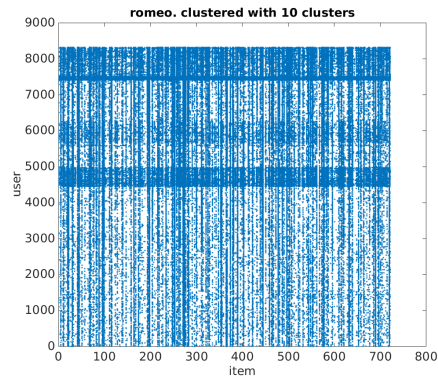


Figure 5.18: *romeo*

Although it is possible to discern some structure from the clustering plots, it is not so clear. The following plots do the same clustering on the user level, but they also sort the items with the most popular item to the left.

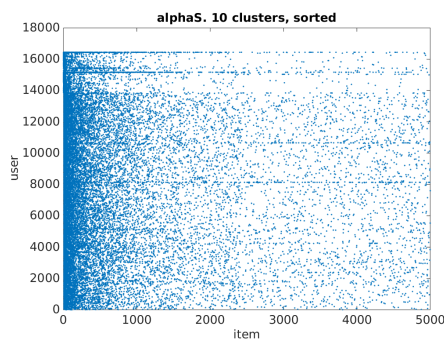


Figure 5.19: *alphaS*

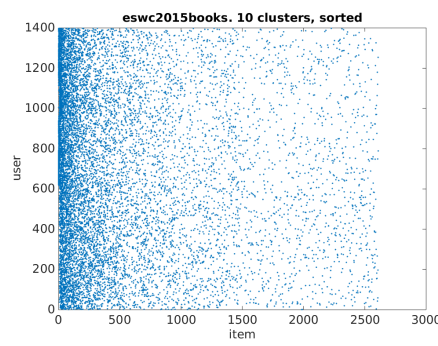


Figure 5.20: *eswc2015books*

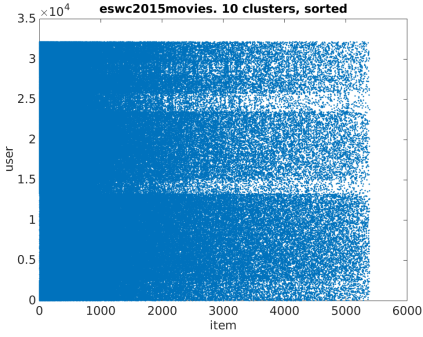


Figure 5.21: *eswc2015movies*

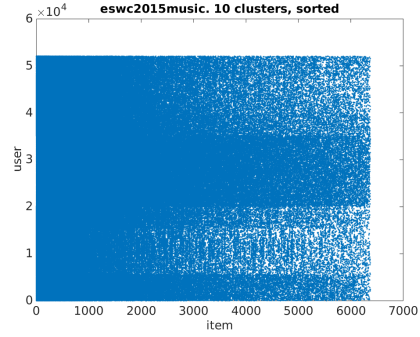


Figure 5.22: *eswc2015music*

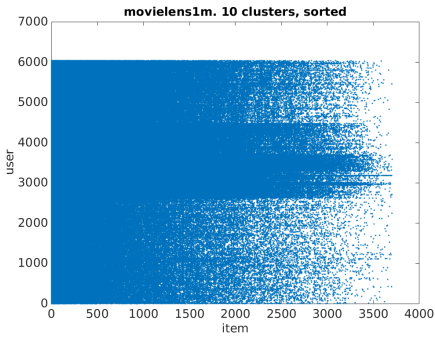


Figure 5.23: *movielens1m*

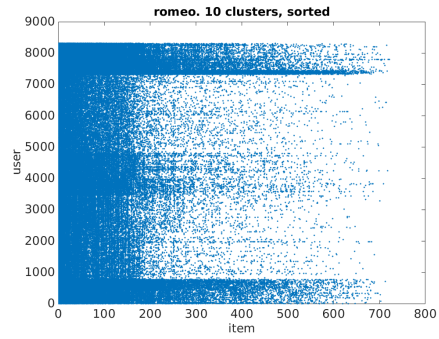


Figure 5.24: *romeo*

With the items sorted the clustering on the user level are more visible. No clear clusters can be seen in either *alphaS* or *eswc2015books* and some grouping can be seen in the other datasets, but clusters cannot be clearly identified. The problem is if the apparent clusters share any items, which is not clear from these plots.

What can be said is that there are some popular items with many interactions and there are items with few interactions. Similarly users with more interactions, which appear to be grouped together, exists. These are all findings supported by the interaction analysis in section 5.2.

5.3.2 Connectivity using Spectral Clustering

The process for clustering with spectral clustering is as follows ⁷

1. Create an affinity matrix, or an adjacency matrix, A_f
2. Construct a graph Laplacian L from A_f
3. Find eigenvalues and eigenvectors of L
4. Select a subspace of eigenvectors
5. Form clusters in the subspace

In our particular case, the interaction matrix A is defined as rows corresponding to users and columns corresponding to items. The affinity matrix needs to be square, but the interaction matrix does not. There are other more complex ways of creating an affinity matrix, but for this purpose modelling an adjacency matrix is sufficient.

A transformation from the interaction matrix A to the adjacency matrix A_f is made by having both users and items as both row and column indices and mirroring the interactions. A_f will then be a symmetric, square matrix. Equation (Eq 5.1) illustrates an example of the matrix structure and figure 5.25 gives a concrete example for *eswc2015books*.

As there are no connections between two users or between two items there will be two large zero rectangles in the top left and the bottom right.

$$A = \begin{matrix} & \begin{matrix} i_1 & i_2 & i_3 & i_4 \end{matrix} \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{matrix} \Rightarrow A_f = \begin{matrix} & \begin{matrix} u_1 & u_2 & u_3 & i_1 & i_2 & i_3 & i_4 \end{matrix} \\ \begin{matrix} u_1 \\ u_2 \\ u_3 \\ i_1 \\ i_2 \\ i_3 \\ i_4 \end{matrix} & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{matrix} \quad (5.1)$$

There are different kinds of Laplacians, mostly differing in how the normalization is done. The one used here is the Generalized Laplacian L ,

$$L = D^{-1}(D - A_f) \quad (5.2)$$

where D is a diagonal matrix called the degree matrix. Each diagonal element $D_{i,i}$ represent the sum of degrees at each node i , calculated as the sum of row i ,

⁷A clear explanation of spectral clustering is given by charlesmartin14: Spectral Clustering: A quick overview, 2012.
<https://charlesmartin14.wordpress.com/2012/10/09/spectral-clustering/>

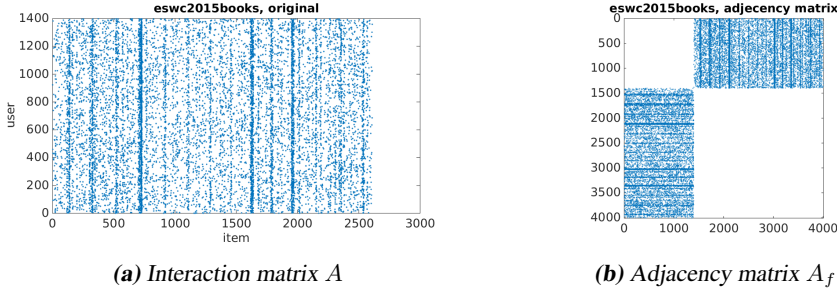


Figure 5.25: This figure illustrates the original interaction matrix 5.25a for *eswc2015books* and the related adjacency matrix 5.25b. Here the matrix A has been moved to the upper right corner of A_f as well as mirrored at the bottom left.

$$D_{i,i} = \sum_j A_{i,j} \quad (5.3)$$

The principal idea is that if good clusters can be identified, then the Laplacian L is approximately block diagonal, with each cluster defined by a block. That is if there are 3 clusters as in (Eq 5.4),

$$L = \begin{pmatrix} L_{1,1} & L_{1,2} & L_{1,3} \\ L_{2,1} & L_{2,2} & L_{2,3} \\ L_{3,1} & L_{3,2} & L_{3,3} \end{pmatrix} \sim \begin{pmatrix} L_{1,1} & 0 & 0 \\ 0 & L_{2,2} & 0 \\ 0 & 0 & L_{3,3} \end{pmatrix} \quad (5.4)$$

then also the lowest eigenvalues and their related eigenvectors correspond to different clusters. In this case the 3 smallest eigenvalues and eigenvector pair would correspond to each cluster, or block, in L .

To be able to identify different clusters, the sorted eigenvalues must have a gap. Figure 5.26 gives a concrete example for *eswc2015books* (another example of clear clusters is figure 5.30 for *alphaS*). It is reasonable to expect there to be clear clusters in the adjacency matrix A_f , as there is a clear gap in the sorted eigenvalues. Note that this doesn't mean that there are clusters in the interaction matrix A , as there is duplicate information in A_f . There might be large areas without interactions, but the indexes might correspond to user-user or item-item which will not have any interactions. But it is a reasonable expectation.

The subspace to find clusters in is some subset of the eigenvectors corresponding to the smallest eigenvalues. The subspace used here is simply the eigenvector corresponding to the 2nd smallest eigenvalue ⁸. Figure 5.27 displays the adjacency matrix A_f for

⁸A practical example which used the same subspace is used as a reference.

Spectral Graph Partitioning and the Laplacian with Matlab, 2006.

<https://www.cs.purdue.edu/homes/dgleich/demos/matlab/spectral/spectral.html>

eswc2015books ordered by the ordering used when sorting the subspace.

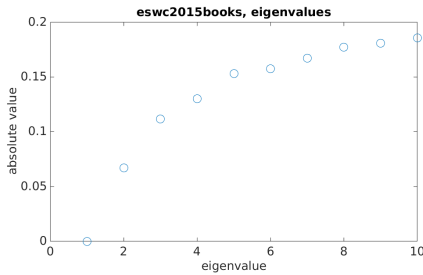


Figure 5.26: The smallest eigenvalues of L for *eswc2015books*.

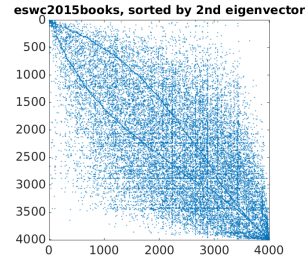


Figure 5.27: Adjacency matrix A of *eswc2015books*, sorted by the ordering used when sorting the 2nd smallest eigenvector of L .

The interpretation of figure 5.26 is that one clear cluster is to be expected due to the gap between the first and second eigenvalue. Some other blurred clusters can be expected as the eigenvalues increase with a decreasing amount. If there are clear clusters the expectation is see square along the diagonal in figure 5.27. There are some horizontal and vertical lines which might be outlines of some triangles, but it is hard to see.

If instead of simply sorting the subspace, *k-means* is used to find a clustering in the subspace, and that ordering is then used to reorder the adjacency matrix. Figure 5.28 displays the adjacency matrix A_f for *eswc2015books*.

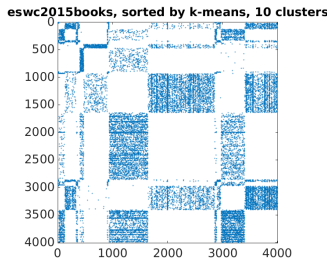


Figure 5.28: Adjacency matrix A_f of *eswc2015books*, subspace clustered with *k-means*.

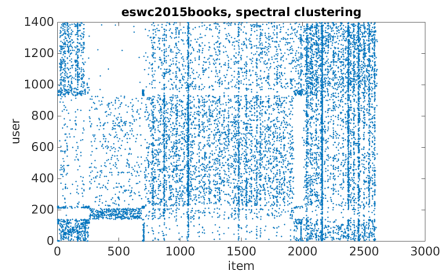


Figure 5.29: Interaction matrix A of *eswc2015books*, reordered using *k-means* clustering information.

The reordered adjacency matrix in figure 5.28 reveal several apparent clusters, but this doesn't directly mean that there are clusters in the dataset as we already know there are large areas without interactions in the adjacency matrix. But using the same ordering to reconstruct the adjacency matrix by reordering both users and items clusters are revealed for *eswc2015books*, shown in figure 5.29.

In contrast with the compactness clustering which only clustered on a user level, this time there is clustering information for both users and items, this is a substantial benefit.

What follows is a similar analysis for the other datasets.

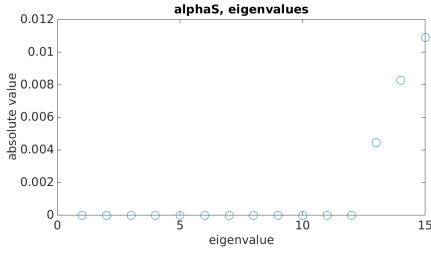


Figure 5.30: The smallest eigenvalues of L for αphaS .

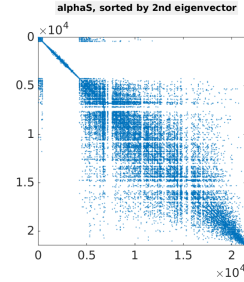


Figure 5.31: Adjacency matrix A of αphaS , sorted by the subspace ordering.

The large number of zero eigenvalues in figure 5.30 suggests that there are many clusters in αphaS . This can also be seen in figure 5.31 where actual clear squares along the diagonal can be seen, as opposed to figure 5.27 for eswc2015books .

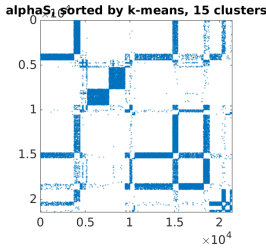


Figure 5.32: Adjacency matrix A_f of αphaS , subspace clustered with k -means.

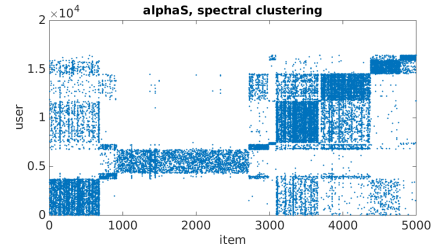


Figure 5.33: Interaction matrix A of αphaS , reordered using k -means clustering information.

The final clustering for αphaS in figure 5.33 reveal that αphaS does have a large number of distinct clusters. From the clustering made here it is possible to identify groups of users with specialized interests, with a subset of appealing items. This could possibly be used to identify personas⁹ in the dataset.

⁹A persona is a description for a class of users or a description of a typical user. This is a requested feature from Comordo's e-commerce clients and spectral clustering could form a base for creating or researching personas for a specific dataset. This process however is not automated and it is just a bi-product of this thesis and not pursued further.

The adjacency matrices for the following datasets are not plotted and only the eigenvalues and the final clustering of the interaction matrix A is presented. They are sufficient to draw conclusions from.

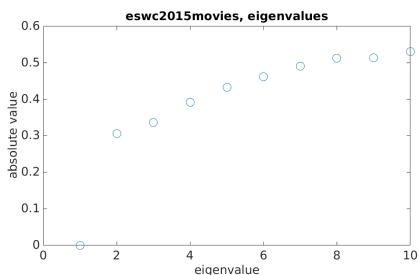


Figure 5.34: The smallest eigenvalues of L for *eswc2015movies*.

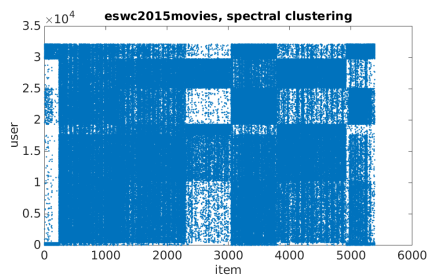


Figure 5.35: Interaction matrix A of *eswc2015movies*, reordered using k -means clustering information.

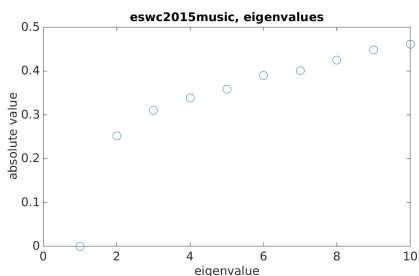


Figure 5.36: The smallest eigenvalues of L for *eswc2015music*.

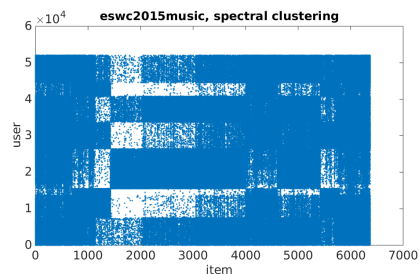


Figure 5.37: Interaction matrix A of *eswc2015music*, reordered using k -means clustering information.

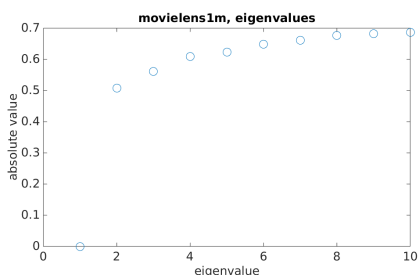


Figure 5.38: The smallest eigenvalues of L for *movielens1m*.

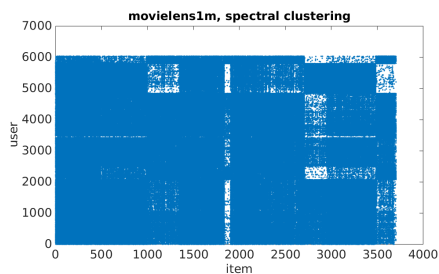


Figure 5.39: Interaction matrix A of *movielens1m*, reordered using k -means clustering information.

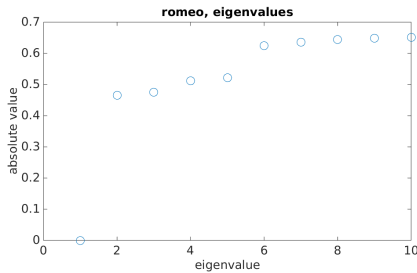


Figure 5.40: The smallest eigenvalues of L for *romeo*.

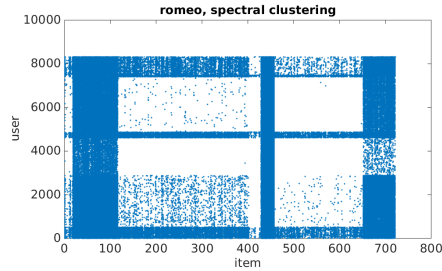


Figure 5.41: Interaction matrix A of *romeo*, reordered using k -means clustering information.

Clusters are visible in the final clustering of the interaction matrix A for all dataset, but with less clear clustering than for *alphaS*. It is supported by their respective eigenvalue plot, with all datasets have a prominent gap but only a single eigenvalue of zero. Figure 5.40 show that there are two other groups of eigenvalues with a gap between them suggesting there might be more prominent clusters in the dataset, which is supported by the clustering in figure 5.41.

The datasets *eswc2015movies*, *eswc2015music* and *movielens1m* also have clusters but they are messier than that of *romeo* or *alphaS*. Part of the reason is the visualization technique where a sparse interaction matrix will produce a cleaner visualization.

This analysis searched for a fixed number of clusters $k = 10$, with *alphaS* using $k = 15$. It is by no means the optimal number of clusters and there might be more clusters in the datasets and there might fewer but “better fitting” clusters. The point of this analysis is not to cluster the datasets in an optimal way, but to examine if the datasets have any clusters.

It is very hard to make recommendations for a dataset without any similarity between users, a random dataset for example, and the existence of clusters shows that there is certain structure in the dataset which could be used to make predictions. The intuition is that in a well clustered dataset users are tightly coupled with users of similar taste and reduces the noise of outlining interactions and a better approximation of the user’s preferences can be made.

6

Parameter tuning

This chapter presents the second major part of this thesis. The chapter begins with defining the convergence criteria for the algorithms with training curves. Then with learning curves the learning process of the algorithms is demonstrated as functional. Parameter space analysis follow which examines the effect of the algorithms' parameters and the final section compares the algorithms and different optimization strategies for parameter tuning.

The tests were run on an **ASUS U36SG RX049V** laptop running slackware 14.1 linux. These are some specs of the laptop:

CPU Intel Core i5-2450M Processor 2.5 GHz

Graphics card NVIDIA GeForce GT 610M 1GB VRAM

Ram 6GB DDR3

HDD 500GB 5400 rpm

6.1 Training curves

Both *katz-eig* and *link-analysis* are iterative algorithms but the descriptions of the algorithms do not specify how many iterations to run. This section examines the stopping criteria for when to stop the iterations. Training curves which plot the evaluation metric with respect to the number of iterations is useful to see the effect of running more iterations. This is then used to define when the algorithms break their loops.

6.1.1 katz-eig

Each K are different for each dataset, see section 6.4, selected for optimal performance and $\beta = \frac{1}{\|A_{train}\|_2}$ is fixed.

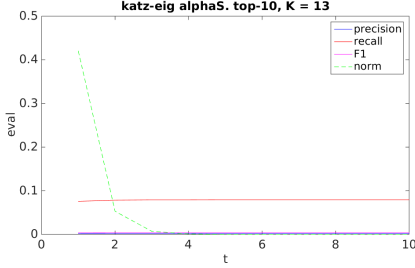


Figure 6.1: *alphaS*

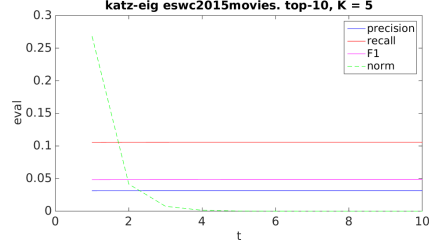


Figure 6.2: *eswc2015movies*

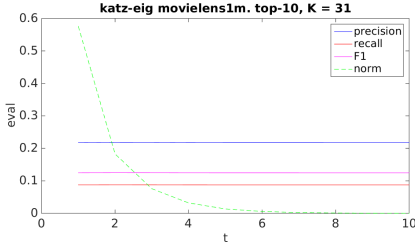


Figure 6.3: *movielens1m*

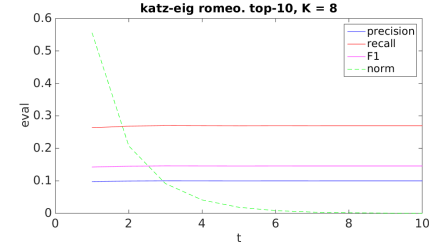


Figure 6.4: *romeo*

The jagged line in the plots represents $\|S_t - S_{t-1}\|_2$, which is a measure of the difference between the current iteration t and the previous iteration. This is made as a measure of the convergence criteria, when $S_t \approx S_{t-1}$ the iterations stops having effect. What can be seen is that S_t converges in relatively few iterations and there is practically no difference in F -measure. It means that more iterations have no real impact.

The convergence criteria is kept and is used to break iterations when $\|S_t - S_{t-1}\|_2 < \epsilon$ with $\epsilon = 0.01$. In practice it means < 10 iterations are done for all datasets. The iteration count could instead be fixed, but the matrix S_t is small and the calculations inside the iteration loop are of low complexity so the convergence is calculated instead of assumed.

In all following usages of *katz-eig*, a value of $\epsilon = 0.01$ is used to break iterations when $\|S_t - S_{t-1}\|_2 < \epsilon$.

6.1.2 link-analysis

Each γ and η are selected for optimal performance with respect to each dataset, see section 6.4.

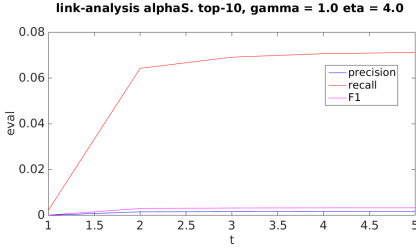


Figure 6.5: *alphaS*

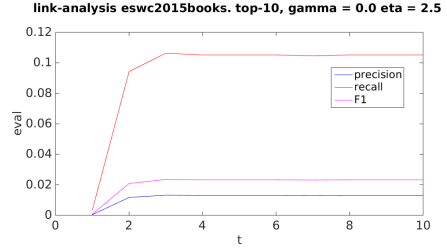


Figure 6.6: *eswc2015books*

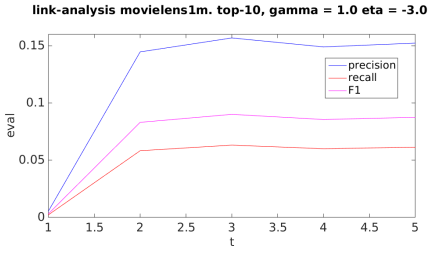


Figure 6.7: *movielens1m*

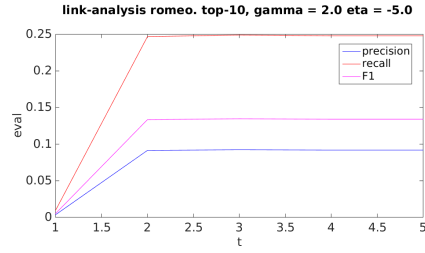


Figure 6.8: *romeo*

For *link-analysis* convergence is also fast. The choice here is to fix t_{max} to a fixed value instead of measuring convergence either by calculating $\|IR_t - IR_{t-1}\|_2$ or by explicitly calculating *F-measure* and measuring the change. This is done because the iteration step in *link-analysis*, in contrast to *katz-eig*, handles large matrices which makes the calculations very time consuming.

In all following usages of *link-analysis*, the iteration count is fixed to $t_{max} = 3$.

6.2 Learning curves

Learning curves plots the evaluation metric with respect to a varying training set size. The expectation here is that the algorithms should fit better the bigger the training set size becomes, given the same test set. The algorithms should produce better recommendations the more data they have to learn from. Learning curves is a good way to see if the learning process work like it is supposed to.

The evaluation uses the training matrix A_{train} and the test set matrix A_{test} . For each step a random selection of a specific size is selected from A_{train} , recommendations are generated and evaluated against the same test set A_{test} . The dimensions of the matrices will be the same, only the number of non-zero elements are increased with the training set size. This is done 10 times for each training set size as to remove variations from the random selection. Ideally more repetitions should be done but due to time constraints they were not.

Optimized parameters as described in section 6.4 are used. The plots also describe the standard deviation.

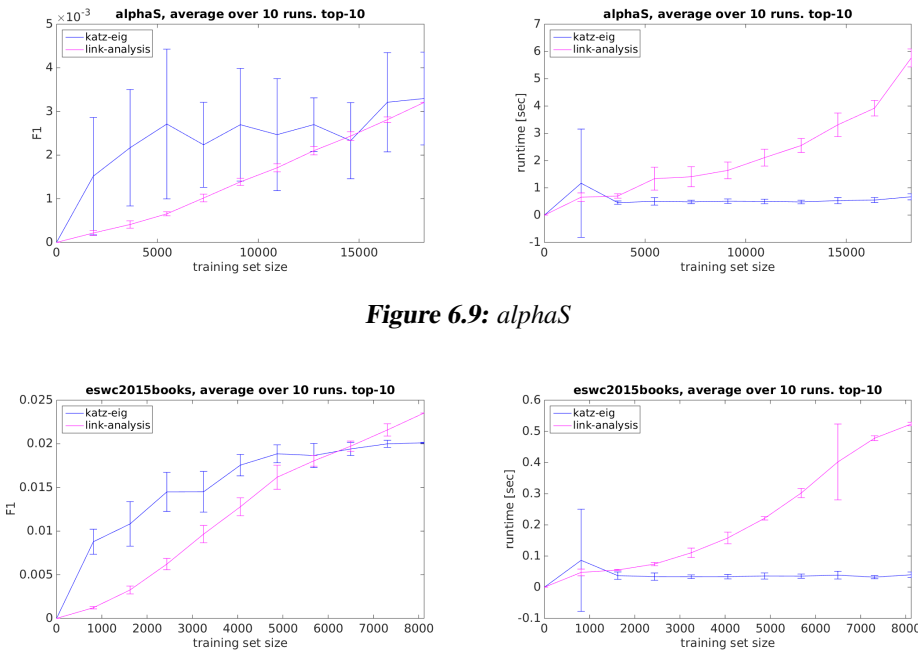
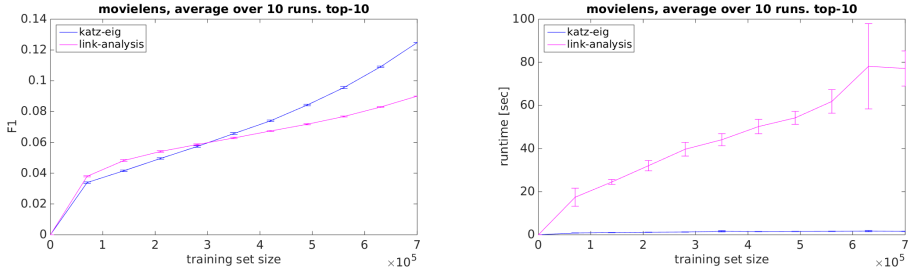
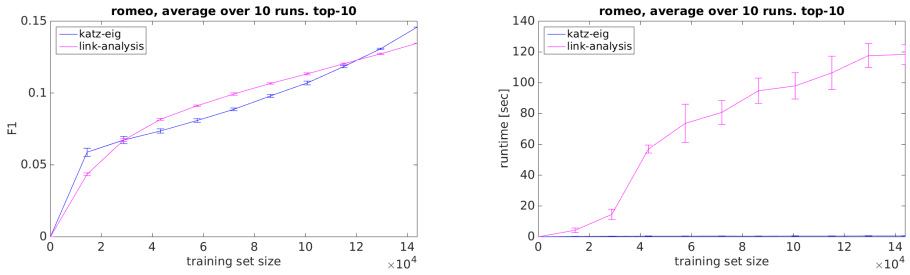


Figure 6.9: *alphaS*

Figure 6.10: *eswc2015books*

Figure 6.11: *movielens1m*Figure 6.12: *romeo*

As the recommendation performance increase with increasing training size the learning process can be said to be working.

A notable observation about the runtime is that for *link-analysis* it increases almost linearly with the increase in training set size, the runtime for *katz-eig* is almost independent of the training set size. This is to be expected as *katz-eig* operates on a low-rank approximation of the interaction matrix A_{train} while *link-analysis* operates directly on the matrix. The sparse matrix format (section 2.1.1) which discards zero elements during calculations is computationally more complex as the sparsity decreases.

Another observation is that *link-analysis* seems to generate better recommendations for the sparse datasets *alphaS* and *eswc2015books* and also for *movielens1m* and *romeo* a short while when the training size was smaller.

The large variation in F -measure for *katz-eig* for *alphaS* and *eswc2015books* can be explained by the sparsity of the datasets. As the training set is randomly selected for each run up to a selected number interactions, it is possible that a large number of users with very few interactions are selected, which makes for a bad basis for recommendations. This is especially true for *alphaS* where most of the users only have one interaction.

The same variation is not seen with *link-analysis* as the recommendations are produced on a link-following basis instead of trying to blur together users like *katz-eig* does.

6.3 Parameter space analysis

This section presents an analysis for what happens to the function space of F -measure with respect to A_{test} when the different parameters for the algorithms are varied.

6.3.1 katz-eig

There are two parameters to *katz-eig*: β , the link diminishing factor and K specifying the K -rank approximation. β is a continous value satisfying $0 < \beta \leq \frac{1}{\|A_{train}\|_2}$. If $\beta = 0$ then the algorithm will only output 0 and if $\beta > \frac{1}{\|A_{train}\|_2}$ the iterations will not converge. $K > 0$ is a discrete value.

What follows is plots over both of the parameters K and β .

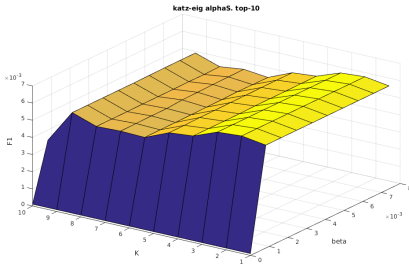


Figure 6.13: *alphaS*

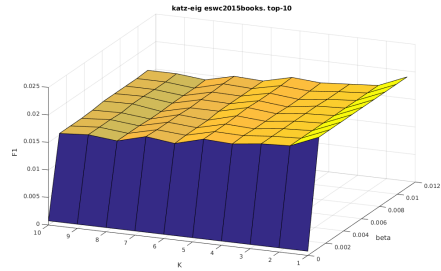


Figure 6.14: *eswc2015books*

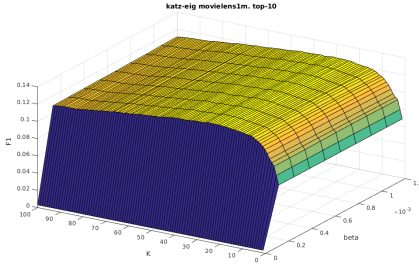


Figure 6.15: *movielens1m*

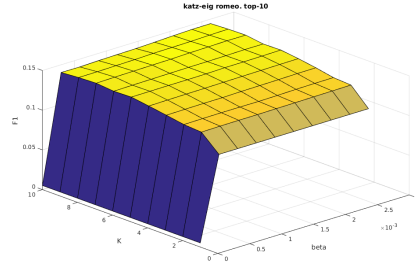


Figure 6.16: *romeo*

It seems like β doesn't have a very big impact on the function value. Some plots with a fixed K follows to better see differences.

The range examined is $0 < \beta \leq \beta_{max} = \frac{1}{\|A_{train}\|_2}$ with a K optimized as described in section 6.4.

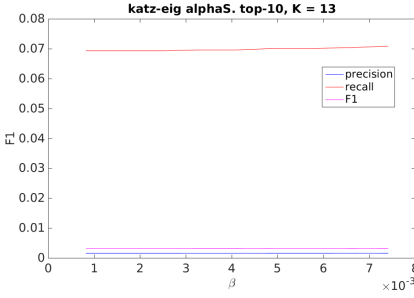


Figure 6.17: *alphaS*. β_{max} is the best value with a 1.9% diff between the minimum and the maximum F1 value.

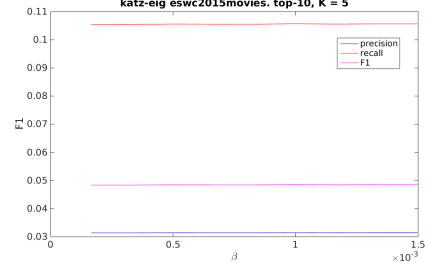


Figure 6.18: *eswc2015movies*. β_{max} is not the best value with a 0.3% diff between the minimum and the maximum F1 value.

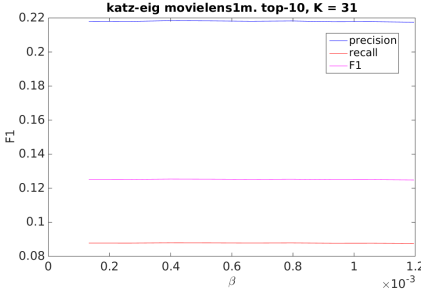


Figure 6.19: *movielens1m*. β_{max} is not the best value with a 0.41% diff between the minimum and the maximum F1 value.

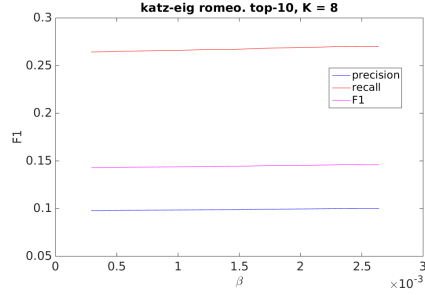


Figure 6.20: *movielens1m*. β_{max} is not the best value with a 2.09% diff between the minimum and the maximum F1 value.

The difference between the optimal β and an arbitrary selected β isn't very large and the difference between the optimal β and β_{max} is even smaller. Table 6.1 is a summary of the evaluated values.

dataset	diff between β_{opt} and β_{max}	diff between f_{min} and f_{max}
<i>alphaS</i>	0 %	2.0 %
<i>eswc2015books</i>	0 %	0%
<i>eswc2015movies</i>	0.039 %	0.28 %
<i>movielens1m</i>	0.41 %	0.41 %
<i>romeo</i>	0.072 %	2.1 %

Table 6.1: A summary of evaluating different β . $\beta_{max} = \frac{1}{\|A_{train}\|_2}$ is the maximally examined β and β_{opt} is the optimal β found in the range $0 < \beta \leq \beta_{max}$. K is individually optimized for the different datasets. f_{min} and f_{max} are the minimal and maximal F1 values obtained.

The K -rank approximation represents different available models for *katz-eig*. The following plots show different values of K . $\beta = \frac{1}{\|A_{train}\|_2}$ for all datasets as the difference between it and the optimal value is negligible. K_m is the value of K which gives the best F -measure for each dataset.

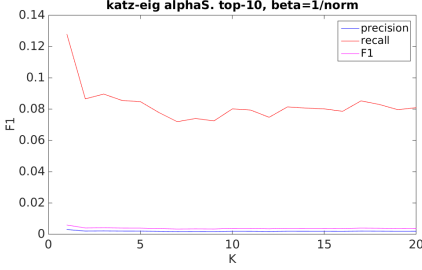


Figure 6.21: *alphaS* $K_m = 13$

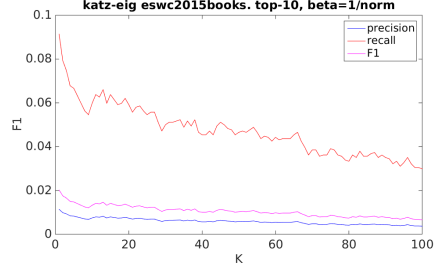


Figure 6.22: *eswc2015books* $K_m = 1$

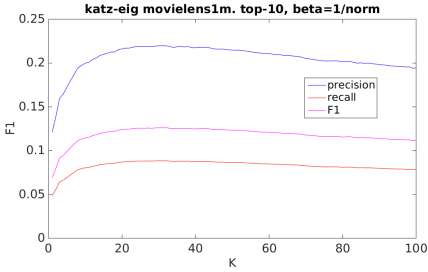


Figure 6.23: *movielens1m* $K_m = 31$

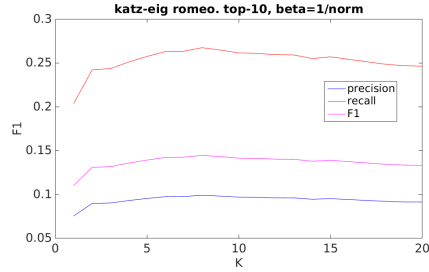


Figure 6.24: *romeo* $K_m = 8$

The function space with respect to K is fairly smooth if not entirely convex. *eswc2015books* is an outlier with a low optimal value $K = 1$ and many local optima. The other datasets display more smooth functions, but there are clear local optima with both *alphaS* and *romeo*.

6.3.2 link-analysis

There are two parameters to *link-analysis*: γ and η , they are both continuous. At $\eta = 0$ all recommendations will always be 0.

The following plot is evaluated using *F-measure* over the parameter space of both γ and η .

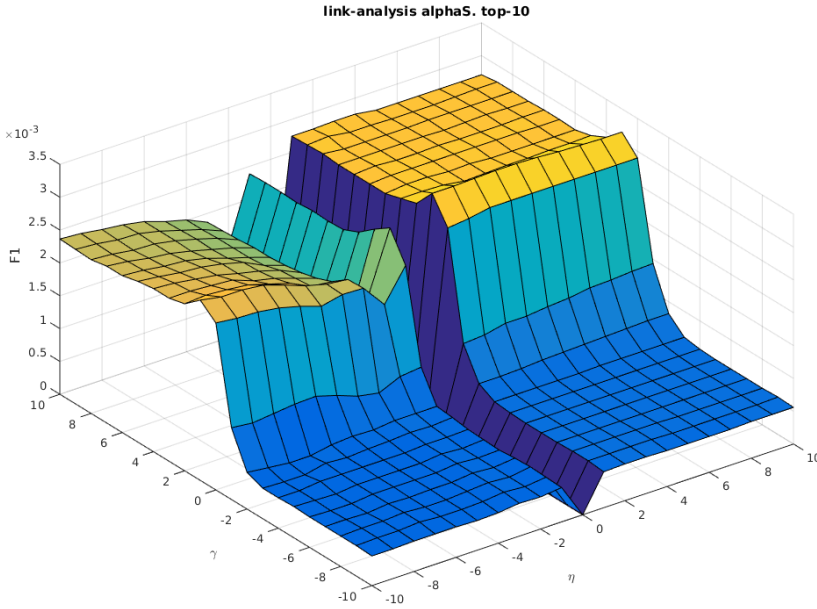
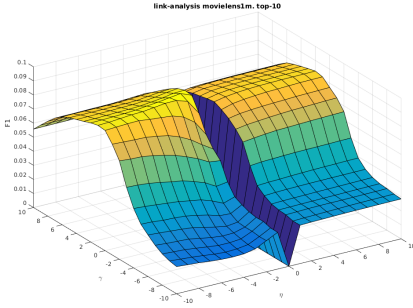
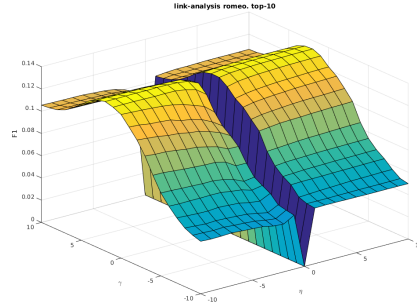
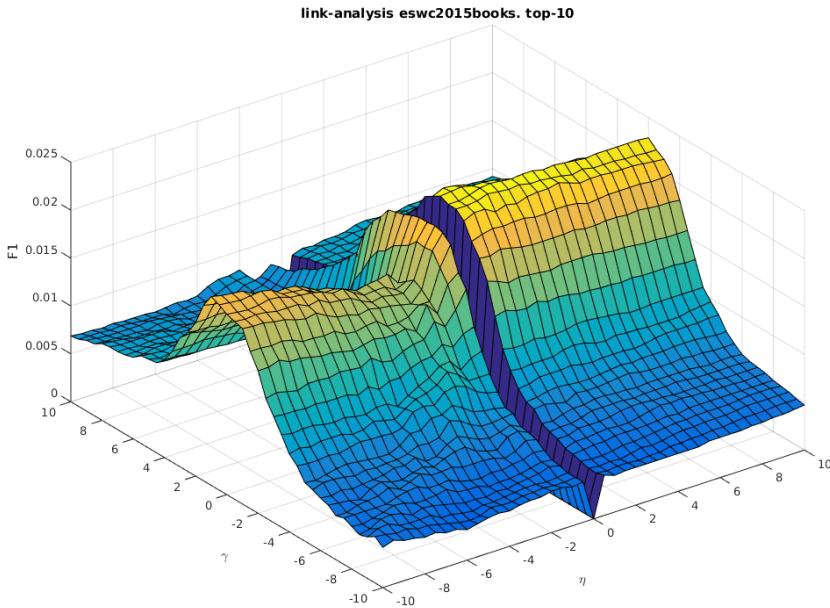


Figure 6.25: *alphaS*

For *alphaS* it appears $\gamma < 0$ is a very bad choice. Both $\eta > 0$ and $\eta < 0$ seems to be fair choices, with $\eta > 0$ being slightly better. As long as $\eta > 0$, the specific choice of η doesn't seem to matter that much. Curiously $\gamma = 0$ represents a peak. There's an anomaly at $\eta = -2$ which is markedly worse than $\eta = -1$ and $\eta = -3$.

**Figure 6.26:** *movielens1m***Figure 6.27:** *romeo*

Similarly for *movielens1m* and *romeo*, $\gamma > 0$ is generally better than $\gamma < 0$. This time $\gamma = 0$ does not represent the maximum function value. The function space seems to be fairly smooth, almost convex, except for $\eta = 0$. For *movielens1m* $\eta < 0$ is the better choice.

**Figure 6.28:** *eswc2015books*

A closer look at *eswc2015books* reveals that the function space isn't as smooth as it might have seemed in the previous plots, several local optima can be seen.

Interestingly some of the datasets have their maximum at $\eta > 0$ but others have $\eta < 0$. What follows is some plots over η and a fixed $\gamma = 1$. The value of $\gamma = 1$ is chosen as a fixed point as the variation of η at that interval seems like a good point to examine for variations, as seen in the previous 3D plots.

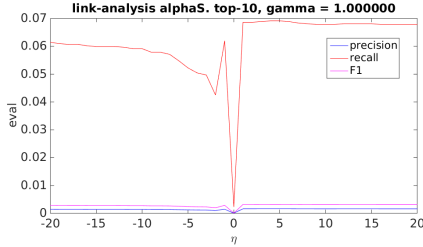


Figure 6.29: *alphaS*

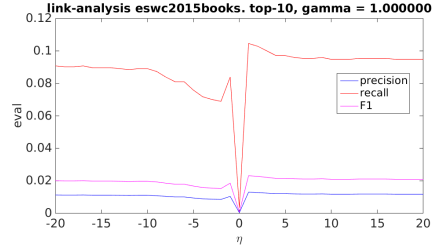


Figure 6.30: *eswc2015books*

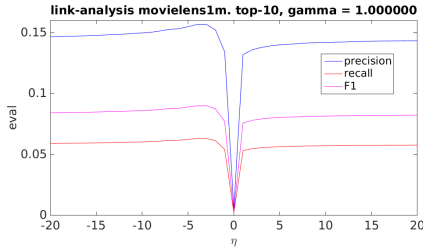


Figure 6.31: *movielens1m*

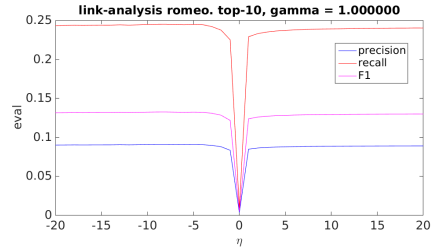


Figure 6.32: *romeo*

Here *alphaS* and *eswc2015books* have maximum *F-measure* with $\eta > 0$ but *movielens1m* and *romeo* with $\eta < 0$. Other than the choice of which sign, the actual value does not seem to matter that much, with some local minimas around $\eta = -2$ for *alphaS* and *eswc2015books*.

γ is the parameter which seems to vary the function value the most. The following plots varies γ while holding $\eta = 1$ constant. $\eta = 1$ is chosen as it appears to be universally good value according to the previous 3D plots.

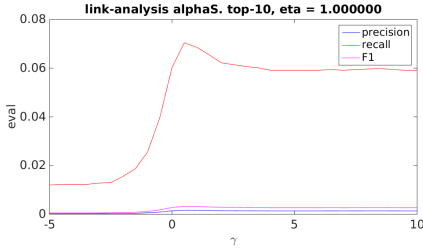


Figure 6.33: *alphaS*

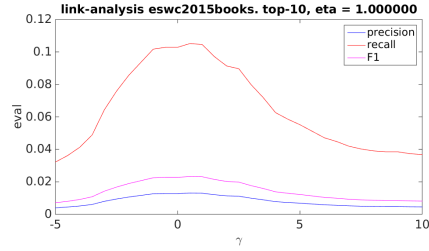


Figure 6.34: *eswc2015books*

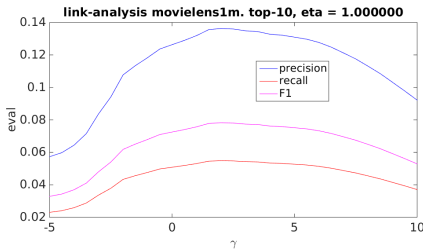


Figure 6.35: *movielens1m*

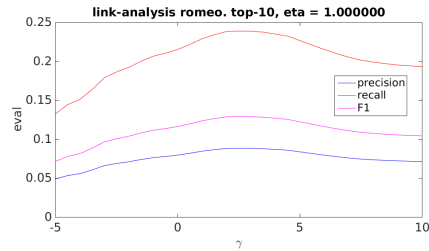


Figure 6.36: *romeo*

Generally $\gamma > 0$ seems like a better choice than $\gamma < 0$. Overall the function space looks fairly smooth with a nice hill shape being visible in all plots. No other local optimas appear to be visible.

6.4 Optimized parameters

This section summarises the optimized parameter values to get good values of F -measure on the corresponding dataset. These values are used throughout the thesis when parameters which would yield good results are needed.

dataset	K
<i>alphaS</i>	13
<i>eswc2015books</i>	1
<i>eswc2015movies</i>	5
<i>eswc2015music</i>	4
<i>movielens1m</i>	31
<i>romeo</i>	8

Table 6.2: Optimized parameters for katz-eig. $\beta = \|A_{train}\|_2$. Found using grid search over $1 \leq K \leq 100$.

dataset	γ	η
<i>alphaS</i>	1	4
<i>eswc2015books</i>	0	2.5
<i>movielens1m</i>	1	-3
<i>romeo</i>	2	-5

Table 6.3: Optimized parameters for link-analysis. Found using grid search over $-10 \leq \gamma \leq 10$ and $-10 \leq \eta \leq 10$ with a step size of 1 (or 0.5 for *eswc2015books*).

6.5 Algorithm comparison

This section starts out with comparing different optimizations strategies for parameter tuning with regards to recommendation quality and speed for *katz-eig* and *link-analysis*. The section is concluded with a comparison of the algorithms against each other using the best evaluated strategies.

6.5.1 katz-eig

The strategies for optimizing algorithm parameters for different datasets, or parameter tuning, has focuses on keeping β fixed and searching for K which optimizes *F-measure*. The analysis in section 6.3.1 shows that varying β has a negligible effect on the algorithm's performance, so β is fixed as $\beta = \frac{1}{\|A_{train}\|_2}$.

What follows is a description of the different optimization strategies evaluated:

grid Does a *grid search* over K , with a fixed step size. β is fixed. $1 \leq K \leq 50$ is examined and a step size of 1 is used.

rand *Random search* over a subspace of K with a fixed β . Depends on the size of the subset to sample over and the number of samples. $1 \leq K \leq 50$ is examined and 12 random samples are used meaning 24% of the subspace is examined.

hill A *hill climbing* algorithm using steepest descent which examines the neighbours of K and moves to the best neighbour, will find a local optima. Uses a fixed β and a step size of 1 is used.

stoch-hill A type of *stochastic hill climbing* algorithm which does a random restart whenever a local optima is found. Also randomly jumps to a random K by a 10% probability whenever a step is taken. Depends on the random jump probability, the subspace of K and the number of iterations. A step size of 1 is used, the space restriction is $1 \leq K \leq 50$ and 12 samples are used meaning 24% of the subspace is examined. The algorithm never revisits an old value.

The following plots compares the recommendation quality and runtime of the different optimization strategies.

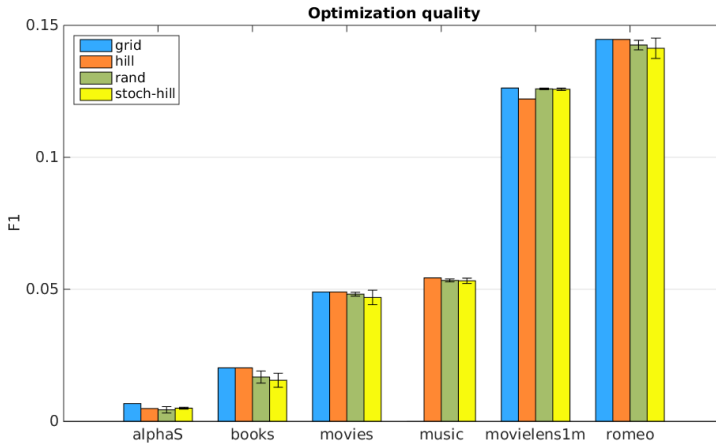


Figure 6.37: Comparison of the recommendation quality given from the parameters found by the different optimization strategies for *katz-eig*.

Figure 6.37 show that all the evaluated strategies generate recommendations of a similar quality. The randomized algorithms **rand** and **stoch-hill** generally generate slightly worse recommendations due to the variance.

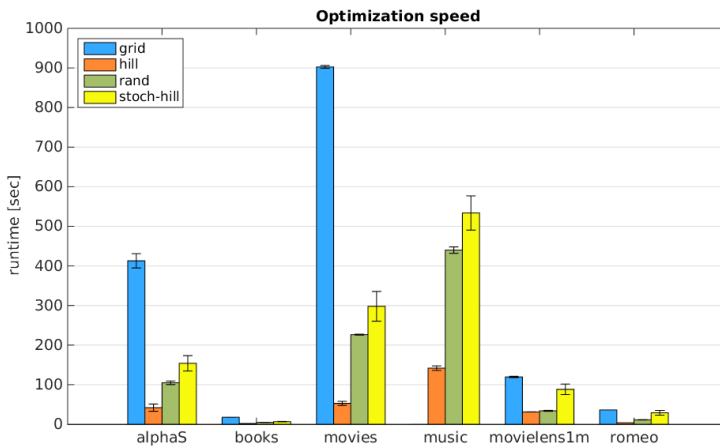


Figure 6.38: Comparison of the runtime of the different optimization strategies for *katz-eig*, given the optimized parameters specified in section 6.4.

The runtime is very poor for the grid based approach as expected. The random algorithms also have poor performance, but this could be corrected by reducing the maximum number of samples they try. Reducing them might reduce the recommendation quality which

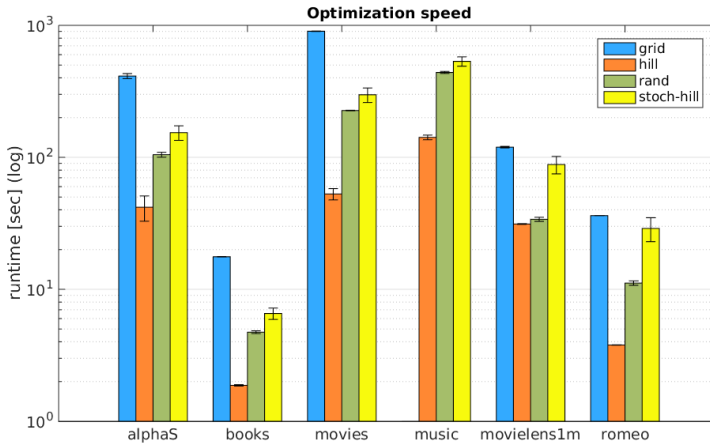


Figure 6.39: Comparison of the runtime of the different optimization strategies for *katz-eig*, given the optimized parameters specified in section 6.4. In a log scale.

already is worse than that of the regular hill climbing algorithm.

These tests show that the regular hill climbing algorithm is the best optimization strategy for *katz-eig* producing similar recommendation quality to the full grid search while being much faster than the other alternatives.

6.5.2 link-analysis

Neither *eswc2015movies* nor *eswc2015music* was runnable with *link-analysis* on the test setup so they are excluded from this discussion.

What follows is a description of the different optimization strategies evaluated:

grid Does a *grid search* over a subset of γ and η , given a fixed step size. $-10 \leq \gamma \leq 10$ x $-10 \leq \eta \leq 10$ was examined with a step size of 1.

adapt-hill An *adaptive hill climbing* algorithm over both γ and η . It starts with a fixed step size and compares the neighbours. If a local optima is found, the step size is decreased. Continues until a specified function, eta or gamma tolerance has been reached. An initial step size of 1 is used and a η tolerance of 1 and a γ tolerance of 0.5 is used. The algorithm aborts whenever a function tolerance of < 0.01 is reached.

adapt-hill-gamma Similar to **adapt-hill**, but instead of searching over η it keeps $\eta = 1$ and optimizes over γ . When a local optima is found, $\eta = -1$ is tried.

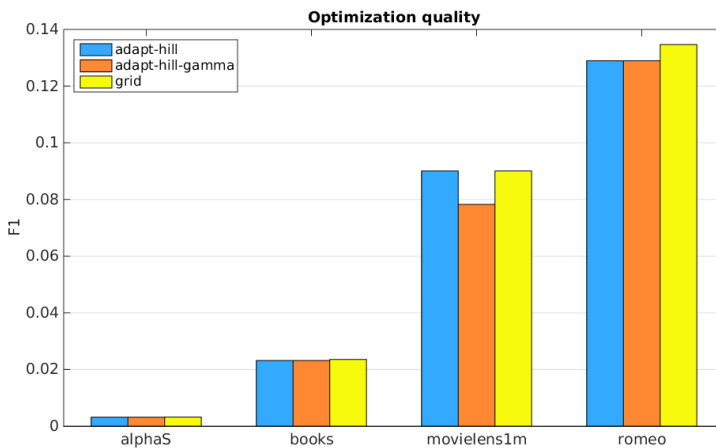


Figure 6.40: Comparison of the recommendation quality given from the parameters found by the different optimization strategies for *link-analysis*.

The only difference between **adapt-hill** and **adapt-hill-gamma** can be seen with *movielens1m*. Both display similar recommendation quality as the full grid search.

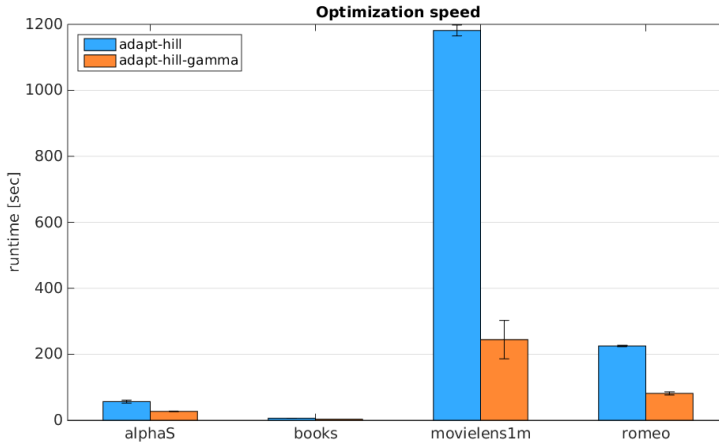


Figure 6.41: Comparison of the runtime of the different optimization strategies for link-analysis, given the optimized parameters specified in section 6.4.

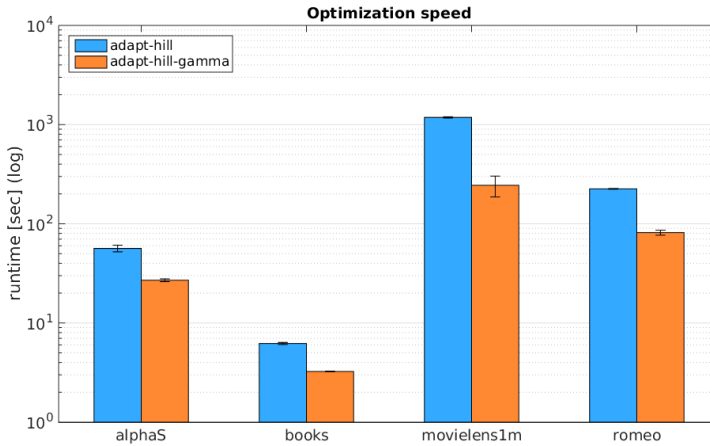


Figure 6.42: Comparison of the runtime of the different optimization strategies for link-analysis, given the optimized parameters specified in section 6.4. In a log scale.

Grid search isn't shown in the runtime plots as it is too slow, for *movielens1m* a runtime of approximately 6 hours is needed. The runtime difference between optimizing over both γ and η compared to only optimizing over γ is large while the difference in recommendation quality is negligible. This shows that purely optimizing over γ using a hill climbing algorithm is better than the alternatives.

6.5.3 Result

This presents a comparison between the best performing optimization strategies for *katz-eig* and *link-analysis* and compares the algorithms against each other. The optimized parameters given in section 6.4 summarises the best performing parameters found for the different datasets. These are the parameters used for the comparison.

Firstly table 6.4 displays the runtime for the algorithms to produce recommendations using the full interaction matrix A . The same results is visualized in figure 6.43.

	katz-eig		link-analysis	
<i>alphaS</i>	3.434409 s	1.038829	10.307018 s	0.068634
<i>eswc2015books</i>	0.027989 s	0.004293	0.720686 s	0.016339
<i>eswc2015movies</i>	1.382029 s	0.516727	x	x
<i>eswc2015music</i>	2.001985 s	0.159226	x	x
<i>movielens1m</i>	1.410586 s	0.107071	154.404908 s	18.179708
<i>romeo</i>	0.216897 s	0.032052	40.477119 s	0.114363

Table 6.4: Runtime of the algorithms using the full interaction history using the optimized parameters given in section 6.4.

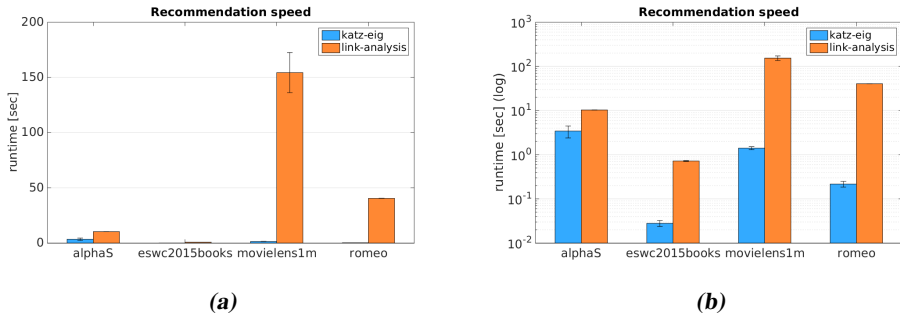


Figure 6.43: Comparison of the runtime of generating recommendations for the full dataset for the different algorithms. (b) shows the speed with logarithmic y-axis. Both plots include the standard deviation of the 10 runs made.

Conclusively *katz-eig* is far superior speed wise. Only *alphaS* is somewhat close. The difference for *movielens1m* and *romeo* is extreme. The bad complexity for matrix multiplication can be blamed as both datasets are larger than the rest and less sparse and *link-analysis* does several full matrix multiplications but *katz-eig* does not.

The time for optimizing the parameters is shown in table 6.5 and visualized figure 6.44.

	katz-eig		link-analysis	
<i>alphaS</i>	41.876401	9.106161	26.978338	0.877453
<i>eswc2015books</i>	1.871681	0.023562	3.245297	0.026975
<i>eswc2015movies</i>	52.749416	5.194185	x	x
<i>eswc2015music</i>	141.664568	5.769257	x	x
<i>movielens1m</i>	31.249456	0.190485	244.484291	58.205089
<i>romeo</i>	3.781222	0.012321	81.478788	4.675901

Table 6.5: Runtime for optimizing the different algorithms for the different datasets. Uses the fastest optimization strategy.

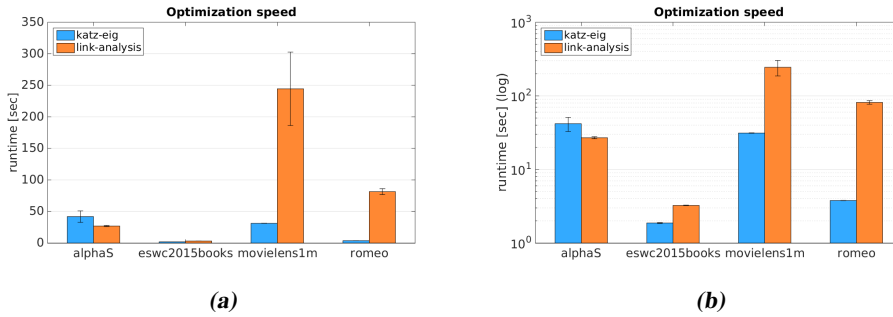


Figure 6.44: The runtime for parameter optimization for the different algorithms. (b) shows the speed with logarithmic y-axis. Both plots include the standard deviation of the 10 runs made.

Again *katz-eig* is superior speed wise, except for *alphaS* where it is slightly slower. The speed difference between the parameter optimization for the algorithms is less than it is for generating the final recommendations. Optimizing for *eswc2015books* is almost as fast for *link-analysis* as for *katz-eig* but the difference is still large for *movielens1m* and *romeo*.

Finally the recommendation quality of the algorithms is given in table 6.6 and visualized in figure 6.45.

	katz-eig	link-analysis
<i>alphaS</i>	0.002174	0.003208
<i>eswc2015books</i>	0.020102	0.023537
<i>eswc2015movies</i>	0.048456	x
<i>eswc2015music</i>	0.054448	x
<i>movielens1m</i>	0.124839	0.090044
<i>romeo</i>	0.145914	0.134655

Table 6.6: Performance on *F*-measure of the different algorithms using the optimized parameters given in section 6.4.

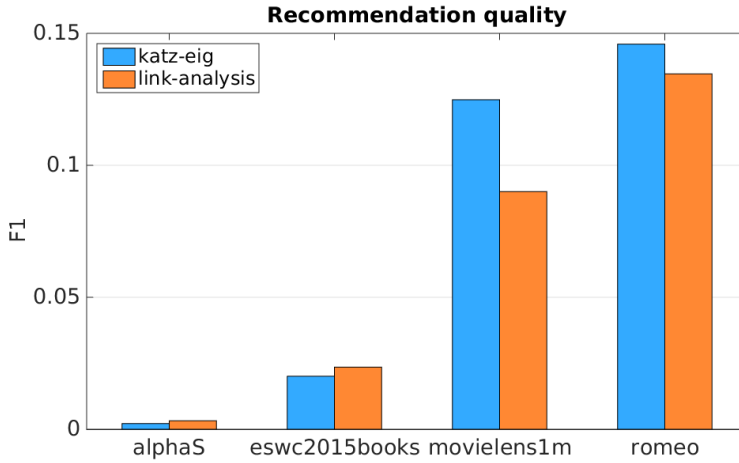


Figure 6.45: Comparison of the recommendation quality given from the different algorithms, given the optimized parameters specified in section 6.4.

For *alphaS* and *eswc2015books* *link-analysis* generates slightly better recommendations compared to *katz-eig* but the difference is very small. For *movielens1m* and *romeo* *katz-eig* provides better recommendations.

Conclusively *katz-eig* provides overall similar or better recommendations and is much faster than *link-analysis*. For sparse matrices *link-analysis* is comparable but for larger and less sparse matrices *katz-eig* is superior.

7

Discussion

This chapter discusses the thesis and future work. The chapter starts by discussing recommender systems in general and the recommender system built for Comordo and gives some possible directions for future work on Comordo's recommender system. After that comes a discussion of the used datasets with comments about some surprising results for specific datasets. Discussion around the evaluation method for recommendation quality follow with thoughts about other metrics to try in future work follow. The chapter concludes with discussing parameter tuning for the different algorithms with comments about some optimization strategies to be tried in future work.

7.1 Recommender systems

One of the purposes of this thesis is to lay the foundation of Comordo Technologies' recommender system. A guiding question in section 1.2.1 asked how a recommender system could be designed to allow for easily extendible input- and output handling.

This question is answered by the plugin system described in section 4.5. A dynamic plugin system designed in this way is very flexible and it can accommodate different formats with little programming effort. Input in a standard form is also easy to handle, for example an interaction matrix already created and stored in a ".mat" data file, as a standard plugin can handle such a case.

It is not enough to only have support for a standard format as Comordo's e-commerce clients use different formats and different types of data. It is possible multiple files in a non-standard format or some preprocessing are needed to handle data from a client. This makes some customization unavoidable and the plugin system allows for this customization.

The recommender system developed in this thesis is very bare bones compared to some larger commercial examples. It is reasonable as the purpose is to produce a first prototype, or a core, for Comordo Technologies to build upon and extend.

7.1.1 Future work

The list of features of a modern recommender system could have is large. An important feature is the act of explaining recommendations, which can be seen in both Netflix and Spotify. Netflix notes that explanations improve trusts in the recommender system and makes the recommendations more effective ¹. This is also supported by the literature which notes the importance of recommendations [1, 20]. The intuition is that if the recommendations come with a note “Recommended to you because you watched *Die Hard*” and if you liked *Die Hard* you’re more likely to value the recommendations.

Extending Comordo’s recommender system with explanations would be a worthwhile direction for future work. Developing algorithms which examine the linked structure could be a possible way of generating explanations, this approach would be especially suited for *link-analysis* but it might also be possible for *katz-eig*. Using a different set of recommendation algorithms with support for explanations [1] might be another possible approach.

Another feature which has gained some popularity lately [2] is diversifying the recommendations. For Netflix it is common that a user account not just represents a single person but a whole household which is why Netflix values diversity in their recommendations ². Diversity is the act of recommending different types of items, in the Netflix example instead of only recommending action movies the recommendations include different genres like drama, animated or horror. This isn’t just relevant when several persons share one account but it also accommodates for different moods of a single person.

Including considerations for diversity into Comordo’s recommender system is another good direction for the future. It is also something Comordo’s clients have shown interest in.

The distinction between offline and online recommender systems is something not considered in the recommender system design by this thesis, but it is an important one. Instead of generating recommendations on a fixed schedule, like once a day, it is desirable to generate recommendations in real time. The way Netflix does it ³ is to use faster and simpler algorithms for generating recommendations in real time and more complex algorithms for offline recommendations which take longer to run. As the complexity of the system grows the line between offline and online algorithms become more blurred and more layers can be added. Netflix has an intermediate layer between the online and offline for example.

When Comordo finalizes their remote API the online/offline distinction should be thought about. I imagine functionality for generating recommendations in real time, so a consumer

¹Netflix: Recommendations beyond 5 stars (Part 1), 2012. <http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html>

²Netflix: Recommendations beyond 5 stars (Part 1), 2012. <http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html>

³Netflix: System Architectures for Personalization and Recommendation, 2013. <http://techblog.netflix.com/2013/03/system-architectures-for.html>

at the e-commerce website can get instant feedback, would be appreciated by their clients.

7.2 Datasets

At the start of this thesis Comordo did not have access to the client specific datasets *alpha*, *alpha*, *alphaS* and *romeo*, they came later during the work. They are interesting because they are real datasets from Comordo's actual clients which makes them commercially interesting to study. Before acquiring these datasets *movielens1m* was the primary dataset used. It was chosen as it is a real life dataset with many interactions used in other work [21]. The datasets *eswc2015books*, *eswc2015movies* and *eswc2015music* are interesting to examine as they form the basis of a recent recommender systems challenge ⁴. They were included to increase the number of datasets in the study.

The datasets *alpha* and *alpha2* are too large to be run on my test setup due to low memory which is why *alphaS* is used instead. Also *eswc2015movies* and *eswc2015music* are not runnable with *link-analysis* as Matlab crashed for unknown reasons when attempts were made. My suspicion is on memory problems there as well. A better setup will be needed to be able to process datasets such as these, or a filtering of the datasets to smaller size would be needed. Optimization of the algorithms with respect to memory usage outside the scope of this thesis but improvements could be made.

The datasets are examined both with regards to the number of interactions and examined if there are any clusters. *alphaS* is an outlier as it had many users (51%) with only one interaction. To generate recommendations a list of popular items could be provided and the users could be removed from the training and test sets for a general speedup and keeping the relative recommendation quality as high or possibly higher. The many one-item users explains the requirement for 61% of all users to be able to reach 95% of the users in table 5.2.

The sparsity of data in *alphaS* explains the better recommendation accuracy for *link-analysis* compared to *katz-eig* (table 6.6) as *link-analysis* is specifically designed to solve the sparse data problem [8, 9]. Similar results can be seen for *eswc2015books* and in the learning curves for *movielens1m* and *romeo* which supports the claim in the literature.

The optimal parameters for *eswc2015books* are fairly strange. For *katz-eig* $K = 1$ gives the best recommendation quality, but that means the recommendations are given by a rank-1 approximation. Also for *link-analysis* $\gamma = 0$ gives the best recommendation quality, which means that no penalization of users with many purchases occur. This could be explained by the low variance of the users interaction count. All users have a very similar amount of interactions and the point of γ is to penalize the user representativeness score $UR(u, \hat{u})$ if u had more interactions than \hat{u} . If they have a similar amount, it doesn't matter.

The main motivation for the clustering analysis was to see if *eswc2015books* had any clusters at all, but it turned out it has clusters (see figure 5.29). I don't have a good

⁴2nd Linked Open Data-enabled Recommender Systems Challenge, 2015. <http://sisinflab.poliba.it/events/lod-recsys-challenge-2015/>

explanation for the strange parameter values for *eswc2015books*, a more in-depth look at the dataset and the generated recommendations would be needed.

Apart from defining a general baseline that all examined datasets have clusters, no other conclusions about the clustering could be drawn. The most clustered dataset *alphaS* receives the lowest *F-measure* and the dataset with the least amount of clear clusters *movielens1m* gets a high *F-measure*, but it's not clear if the clustering is the reason. They are endpoints with respect to sparsity with *alphaS* being the most sparse and *movielens1m* the least.

The bad recommendations for *alphaS* can also be explained by it having a very large percentage of users with very few interactions, which per definition will give a bad *F-measure*. Similar reasons can be given for *eswc2015books*. There most users have less than 10 interactions, but as we're generating a top-10 list of recommendations a large part of the recommendations will not be found matched against the test set and will produce a low *F-measure*.

Better recommendations are generated for *movielens1m* and *romeo*, with both algorithms. Both datasets are larger while being less sparse. The defining difference between datasets with good recommendations and datasets with bad seems to be the sparsity level and the number of interactions per user. Which makes intuitive sense as it is easier to predict for users with more interactions than for users with fewer interactions. Clustering does not seem to be a factor for generating good recommendations.

7.3 Evaluation

Different types of evaluation metrics have been used throughout the literature. The one used in this thesis, *F-measure*, is a popular one but other metrics could yield different results. *Diversity* [2] is a metric which measures the diversity of the recommendations. In conjunction with *F-measure*, *Diversity* was the evaluation metric used for the 2nd Linked Open Data-enabled Recommender Systems Challenge⁵. *Rank Score* [9] is another metric designed to rate a ranked list, list with the most recommended items, which is evaluated in thesis. *Rank Score* and *F-measure* is used to evaluate *link-analysis* [9]. For future work it would be interesting to evaluate *link-analysis* and *katz-eig* using these metrics to see if the parameter analysis differ.

Another defining feature is that in this thesis only recommendations for new items are generated. This makes sense for some of Comordo's clients as they are concerned with recommending new products, but other businesses might want to recommend old products as well. Modifying the evaluation process to also include old products might change both the performance and the behaviour of the algorithms. This is something which might be pursued more in future work.

The reasoning for evaluating *F-measure* against a top-10 list of recommendations instead of the whole recommendation matrix with some threshold is also because it's most inter-

⁵Evaluation | 2nd Linked Open Data-enabled Recommender Systems Challenge, 2015.

<http://sisinflab.poliba.it/events/lod-recsys-challenge-2015/tasks/evaluation-service/>

esting for Comordo and their clients. A top 10 list was chosen for simplicity instead of evaluating different number of recommendations which would make the analysis more complex and time consuming.

In the end the reasoning for using *F-measure* is one of simplicity. *Precision*, *Recall* and *F-measure* is in some ways the basic way of evaluating recommendations which can form a good basis and other metrics which measure more specific things (like *Diversity*) can complement and extend that basis. This extension is expanded upon more in section 7.1.1 of future work for Comordo's recommendation system.

7.4 Parameter tuning

The second of the two large parts of the thesis is to analyze and create parameter optimization strategies for *katz-eig* and *link-analysis*. This is specified by the guiding questions in section 1.2.1 and focus is placed on how to generate recommendations in practice.

First a recommender system around the algorithms is built containing all infrastructure necessary to generate recommendations, see section 4.5. Then the analysis of the parameter space with regards to recommendation quality guides the evaluation of suitable optimization strategies which then completes the parameter optimization part of the thesis.

katz-eig generally has much better runtime performance compared to *link-analysis* and generates similar or better recommendation quality. *link-analysis* gives slightly better recommendation quality for sparse datasets. It is expected as the primary motivation for *link-analysis* is to generate recommendations for sparse data, but *katz-eig* still gives comparable recommendations even for these datasets.

Optimization is very expensive, but generating recommendations for the full datasets for both algorithms is fairly fast. Both algorithms can both be useful commercially when the recommendations are generated offline. The only reason to use *link-analysis* over *katz-eig* is if a known dataset is very sparse and fairly small otherwise *katz-eig* is the superior choice.

In general gradient descent strategies and in particular the popular stochastic gradient descent were not evaluated as a suitable cost function, with a gradient, for *katz-eig* or *link-analysis* could not be find.

7.4.1 Parameters of *katz-eig*

The impact of varying β is very small, as shown in section 6.3.1. It is surprising as according to the model β represents the link dampening. This means that fixing β to a constant value while optimizing over K could give speed improvements while retaining recommendation quality. Another surprising result is that the training curves in section 6.1.1 show that the number of iterations does not have a sizable effect on *F-measure*. There is some improvement, but it is a very minor one.

These results suggests that *katz-eig* gives very similar recommendations as directly using a SVD- k approximation would, which is what happens when $\beta = 0$. The number of

iterations intuitively represents how many links are followed, but if the recommendation quality is the same regardless of the number of iterations, then only one link followed is enough. Recommendations based on SVD, or low rank matrix approximation in general, is an acknowledged way of generating good predictions [2, 14] so the assumption is reasonable.

The intuition that a low rank approximation could form the direct basis for recommendations is that the effect of a matrix approximation necessarily blurs rows and columns, meaning the user and item interaction history is blurred leading to users with similar interactions are blurred together.

Varying K however affects *F-measure* a lot more, as seen in section 6.3.1. Although a bit jagged, the plots indicate a “hill” shape which suggests that a hill climbing strategy could be a viable optimization strategy. As seen in section 6.5.1 this does seem to be the case as a hill climbing strategy produce nearly as good a result as a full grid search. A variant with random jumps and restarts is also tried but the standard hill climber outperformed the stochastic version.

7.4.2 Parameters of link-analysis

The paper introducing *link-analysis* [8] does not include η in the description. It is instead introduced by the same authors in a later paper [9] but there it is just set to $\eta = 1$ without further comments or analysis, which is surprising.

The analysis in section 6.3.2 shows that $\eta = 1$ is a well performing value. The modelling of *link-analysis* supports it as the most logical value, which probably is why they chose it. When $\eta > 1$ very similar results as $\eta = 1$ are obtained. This can be explained by the algorithm description. The purpose of $UR_0 = \eta * I_M$ is to keep a high user representativeness score for the user itself. It doesn't matter that much how high the value is, as long as it is higher than the surrounding values. With the normalization step included in the iterations a value of $\eta \geq 1$ makes sure of that. Worth pointing out is that the optimal parameters can have $\eta > 1$, but it is not with a very big margin. Inaccuracy in multiplications is a plausible explanation.

More interesting and harder to explain is values $\eta < 0$, which can be even better than $\eta > 0$. I have found no reasonable explanation and it might be worth investigating further.

The same papers [8, 9] mentions that investigations with $0 \leq \gamma \leq 1$ have been made and then a value of 0.9 was a good value. My investigations include a larger interval and optimal values for γ can be found outside that range, *romeo* has a local maxima with $\gamma = 2$. The findings in this thesis however do support the claim that 0.9 is a reasonably good value.

Analysis of γ in section 6.3.2 suggests that *F-measure* is almost convex with a clear “hill” in the function space. There are small local maxima but as a whole it follows a clear hill form. This suggests that a hill climbing optimization strategy might perform well for these datasets. It might also be worthwhile to keep η fixed as is done in [8, 9].

This is supported by the tests in section 6.5.2 which show that an adaptive hill climbing strategy produce recommendations almost as good as a full blown grid search in only a

fraction of the time needed. The difference in recommendation quality between optimizing over both γ and η compared to just optimizing over γ while $\eta = 1$ or -1 is small, but the runtime difference is not.

7.4.3 Future work

Due to time constraints several optimization techniques are not examined by this thesis. Stochastic hill climbing is examined for *katz-eig* but not for *link-analysis*. It is not a very promising strategy as the parameter space still has a clear hill shape, but depending on the exact implementation it could still be a promising technique. The fixed parameters for the examined algorithms were not optimized and performance improvements could be gained by tweaking them more.

Simulated annealing is a way of randomly sample, similar to the random sampling and stochastic hill climbing investigated for *katz-eig*, and the expectation is that simulated annealing might outperform these strategies. *Bayesian optimization* is another promising optimization strategy to try.

For future work it would be interesting to examine the performance of optimizing on K for *katz-eig* and on γ or on γ and η for *link-analysis* using simulated annealing and bayesian optimization. The central question is if these strategies can outperform the regular hill climbing algorithm for datasets such as these with very prominent hill shapes.

8

Conclusions

A *implicit feedback* recommender system with the recommendation algorithms *katz-eig* and *link-analysis* is built. Optimization strategies for learning the algorithms' parameters and fit them to different datasets are implemented and evaluated.

Recommendation quality is measured using *F-measure* and evaluated on a top 10 list. For sparse datasets *link-analysis* gives slightly better recommendation quality compared to *katz-eig*, but the performance is comparable. For the other datasets *katz-eig* gives better recommendations. Speed wise *katz-eig* is superior. As the difference in recommendation quality for sparse datasets is so small *katz-eig* is the best general choice as the recommendations are better for the other datasets and it is generally much faster. The recommendations are better with datasets which have more interactions and worse for sparse datasets.

For future work *bayesian optimization* and *simulated annealing* could be explored as possibly more efficient optimization strategies. The recommender system could be extended with regards to diversity and ability to explain the recommendations.

This is a summary of the guiding questions posed in section 1.2.1 with answers.

Q: How can a recommender system be designed to allow for easily extendible input- and output handling?

A: With a plugin system where each plugin correspond to a different format. The developed system is described more in section 4.5.1.

Q: How can learning and recommendation using *link-analysis* and *katz-eig* be performed in practice, with regards to speed and recommendation quality? How shall learning and optimization of their parameters be done?

A: Using a recommender system as described in section 4.5 learning and recommending can be accomplished in a general way.

Varying β for *katz-eig* has little to no effect and *link-analysis* varies mostly depending on the sign with respect to η . The best parameter optimization strategy for *katz-eig* is to fix $\beta = \|A_{train}\|_2$ and optimize K using a hill climbing algorithm. It gives similar recommendation quality compared to a full grid search but with better speed. Similarly the best optimization strategy for *link-analysis* only examines $\eta = 1$ and $\eta = -1$ while using an adaptive hill climbing algorithm to optimize γ .

Appendix

A

Code

A.1 ESWC reader plugin

```
class Eswc():

    def add_arguments(self, parser):
        """Parse command line arguments"""

        parser.description = "Load eswc data"
        parser.add_argument('file', metavar='eswc_training_file', type=str
                            , help='The training file')

    def load(self, args):
        """Load users and items."""

        items = {}
        users = {}

        # Each line is composed by: userID \t itemID
        with open(args.file) as f:
            for line in f:
                s = line.rstrip().split('\t')
                user_id, item_id = s[0], s[1]

                if not item_id in items:
                    items[item_id] = Item(item_id)

                if not user_id in users:
                    users[user_id] = User(user_id)

                users[user_id].add_history(item_id, 1)

        return users, items
```

Bibliography

- [1] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. IEEE, 2008. Cited on pages 2, 7, 17, 18, 19, 20, 21, 22, and 72.
- [2] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013. Cited on pages 2, 17, 18, 21, 72, 74, and 76.
- [3] Gábor Takács and Domonkos Tikk. Alternating least squares for personalized ranking. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 83–90. ACM, 2012. Cited on pages 2, 21, and 22.
- [4] Fidel Cacheda, Víctor Carneiro, Diego Fernández, and Vreixo Formoso. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Transactions on the Web (TWEB)*, 5(1):2, 2011. Cited on pages 3 and 28.
- [5] Donghyuk Shin, Si Si, and Inderjit S Dhillon. Multi-scale link prediction. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 215–224. ACM, 2012. Cited on pages 8 and 22.
- [6] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953. Cited on page 8.
- [7] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999. Cited on page 11.
- [8] Zan Huang, Daniel Zeng, and Hsinchun Chen. A link analysis approach to recommendation under sparse data. *AMCIS 2004 Proceedings*, page 239, 2004. Cited on pages 11, 22, 73, and 76.
- [9] Zan Huang, Daniel Zeng, and Hsinchun Chen. A comparison of collaborative-filtering recommendation algorithms for e-commerce. *IEEE Intelligent Systems*, (5):68–78, 2007. Cited on pages 11, 22, 73, 74, and 76.

- [10] Peter Norvig and Stuart Russell. *Artificial Intelligence, A Modern Approach*. Pearson, third edition, 2010. Cited on pages 16, 17, and 20.
- [11] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999. Cited on page 17.
- [12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012. Cited on page 20.
- [13] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012. Cited on page 20.
- [14] Linyuan Lü, Matúš Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, and Tao Zhou. Recommender systems. *Physics Reports*, 519(1):1–49, 2012. Cited on pages 21 and 76.
- [15] James Bennett and Stan Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007. Cited on page 21.
- [16] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009. Cited on page 21.
- [17] Siwei Lai, Yang Liu, Huxiang Gu, Liheng Xu, Kang Liu, Shiming Xiang, Jun Zhao, Rui Diao, Liang Xiang, Hang Li, et al. Hybrid recommendation models for binary user preference prediction problem. In *KDD Cup*, pages 137–151, 2012. Cited on page 21.
- [18] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004. Cited on page 21.
- [19] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461. AUAI Press, 2009. Cited on page 22.
- [20] Nava Tintarev and Judith Masthoff. A survey of explanations in recommender systems. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 801–810. IEEE, 2007. Cited on page 72.
- [21] Niklas Ekvall. Movie recommendation system based on clustered low-rank approximation. 2012. Cited on page 73.

Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: <http://www.ep.liu.se/>