

# **Diplomarbeit**

## **Performant Trust and Similarity Metrics for Inconsistent Knowledge-bases**

*Florian Mittag*

Dezember 2008

Betreuung: Prof. Dr. Andreas Dengel  
Dipl. Inf. Malte Kiesel

Arbeitsgruppe Wissensbasierte Systeme  
Prof. Dr. Andreas Dengel

Technische Universität Kaiserslautern  
Fachbereich Informatik

Florian Mittag  
Konrad-Adenauer-Straße 63  
67663 Kaiserslautern

Kaiserslautern, den 18. Dezember 2008

## **Erklärung**

Hiermit erkläre ich, dass ich die Arbeit selbstständig verfasst und nur die angegebenen Hilfsmittel verwendet habe.

(Florian Mittag)

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Structure of the thesis . . . . .	2
1.3	Prerequisites . . . . .	3
<b>2</b>	<b>Skipforward overview</b>	<b>4</b>
2.1	Skipinions: The top level ontology . . . . .	5
2.2	Formal definitions . . . . .	7
2.3	Maintaining information provenance and authentication . . . . .	9
2.4	Item identity . . . . .	10
2.5	Manual and automatic annotation . . . . .	11
2.6	Architecture . . . . .	11
<b>3</b>	<b>Theoretical Foundations</b>	<b>12</b>
3.1	Trust . . . . .	12
3.1.1	What is trust? . . . . .	13
3.1.2	Personalized views using trust . . . . .	13
3.1.3	Trust models . . . . .	14
3.2	Recommender Systems . . . . .	14
3.2.1	Collaborative Filtering . . . . .	15
3.2.2	Content-based Filtering . . . . .	18
3.2.3	Hybrids . . . . .	19
<b>4</b>	<b>Approach</b>	<b>20</b>
4.1	User similarity . . . . .	20
4.1.1	Adapting the Pearson correlation . . . . .	21
4.1.2	Limits of the Pearson correlation . . . . .	23
4.1.3	The constrained Pearson correlation . . . . .	24
4.1.4	Computational complexity and incremental calculation . . . . .	25
4.2	Creating personalized views . . . . .	29
4.2.1	Confidence in similarities . . . . .	30
4.2.2	Trust model in Skipforward . . . . .	32
4.2.3	Feature aggregation . . . . .	32

4.2.4	Using domain knowledge to reduce data sparsity . . . . .	33
<b>5</b>	<b>Implementation</b>	<b>36</b>
5.1	Main algorithms . . . . .	36
5.1.1	Initializing the similarity values . . . . .	36
5.1.2	Listening for feature changes . . . . .	38
5.1.3	Incremental update of similarity values . . . . .	38
5.1.4	Inferencing . . . . .	41
5.2	Data structures . . . . .	42
5.2.1	Feature Cache . . . . .	42
5.2.2	Incremental user similarities . . . . .	45
<b>6</b>	<b>Evaluation</b>	<b>46</b>
6.1	Evaluation parameters . . . . .	46
6.2	Statistical results . . . . .	46
6.3	Performance . . . . .	54
<b>7</b>	<b>Summary and Outlook</b>	<b>56</b>
7.1	Outlook . . . . .	56
7.1.1	Optimization of data structures . . . . .	56
7.1.2	Relationships between feature types . . . . .	57
7.1.3	Handling of non-binary feature types . . . . .	57
7.1.4	Similarities on partial data sets . . . . .	58
<b>A</b>	<b>Glossary</b>	<b>59</b>
<b>B</b>	<b>Proof of equations</b>	<b>61</b>
<b>C</b>	<b>Feature types of the evaluation</b>	<b>63</b>
	<b>Bibliography</b>	<b>64</b>

# Chapter 1

## Introduction

### 1.1 Motivation

Recently, I was making plans for the evening and decided to go to the cinema. Since I was very busy in the days before that, I had paid little attention to the movies that were shown at that time and had only a few hours to choose which one I wanted to see.

My first source of information about the scheduled movies was the Internet Movie Database (IMDb<sup>1</sup>), where I could view the average ratings of the movies, user comments and other information like cast and runtime, but there were several problems. The ratings are averaged overall opinions that not necessarily reflect the way I would rate those movies. And even if I agreed with the rating of a particular movie in general, I might not be in the mood for it that night. There are also many details about a film that are not contained in the description of the IMDb, e.g., if the stunts are exaggerated or if the humor is silly. These information might be found in user comments, but then again they can be biased by personal opinions and I don't know about my consensus with these.

I then asked some friends for their opinion about aspects of the movie that I knew we often agreed upon. For example, one of my friends almost always thought of stunts to be exaggerated and unrealistic, so I knew that I could ignore his comment about this feature. On the other hand, we shared the same sense of humor, so his opinion about this aspect was a reliable information to me. However, due to the fact that I had less than a hundred friends and did not have the time to ask more than seven, the number of friends I could ask was low compared to the vast amount of people expressing their opinions on the internet.

As can be seen, each source of information has its advantages and disadvantages. The internet provides such a large amount of information on almost everything that it often requires a big effort to find relevant ones. This problem is increased by the large number of collaboratively created knowledge-bases that allow users to create and modify content or add comments to it. This sometimes

---

<sup>1</sup><http://www.imdb.com>

makes it hard to determine the source of information, which has a large influence on the trust we put in it.

The “old-fashioned” way of asking friends and like-minds does not offer as plenty of information as the internet, but our personal knowledge about its source often makes the available information much more useful, because we know how to interpret it.

It would be desirable to have a system that allows users to express their opinions about items, such as movies, books and songs, and supports the search for information by automatically aggregating possibly contradictory opinions into a personalized view. As such, the data has to be represented in a machine-readable format and has to provide provenance information.

Skipforward is a system that is being developed to meet this criteria. It already allows different users to make machine-readable statements about certain items, while maintaining provenance information. However, it is currently not capable of determining the relevance of information based on previous experience. The available information has to be interpreted differently for each user to turn them into personalized information.

The basic idea of this thesis is to automatically calculate similarities between pairs of users for different topics based on previously expressed opinions. These similarities are then used to estimate the relevance of information for a specific user to create a personalized view. Since the number of items and opinions about them will grow during the usage of the system, the algorithms for this calculation are designed to work incrementally and still perform efficiently on large knowledge-bases.

## **1.2 Structure of the thesis**

Chapter 2 introduces Skipforward, a framework for distributed, collaborative, and personalized annotation of resources with structured metadata. The format of stored data will be described, as it is the foundation for the later examination of similarity measures and for the algorithms developed.

In Chapter 3, formal definitions of trust are discussed, as they can be used to describe trustworthiness of information and their sources. After that, a short overview on recommender systems is given and important terms and algorithms are defined and explained.

Based on this knowledge, Chapter 4 elaborates different algorithms to calculate user similarities with regard to performance and usefulness of their results. The aggregation of inconsistent information into a personalized view is then realized using a formal trust model.

The integration of the developed components into the Skipforward system is described in Chapter 5. To ensure the performance of the algorithms, certain requirements have to be made to the underlying data structures.

An evaluation of the implementation with real data concerning music annota-

tion is presented in Chapter 6. At first, it is explained how the data was gathered and what difficulties arise from the nature of its domain. Based on this data the results of the developed algorithms are examined with respect to their plausibility and time performance.

Chapter 7 summarizes the results of this thesis and gives an outlook on further work and research that has to be done for this topic.

### **1.3 Prerequisites**

The reader should be familiar with the idea of the Semantic Web and in particular with RDF/S<sup>2</sup> and URIs. Although it is not necessary to be familiar the Web 2.0 and tagging, experience with collaborative platforms like Wikis, interactive online databases and recommendation services is helpful in order to get a full grasp of the problems and methods of resolution presented in this thesis.

Skipforward and the algorithms described in this thesis are implemented in Java 5<sup>3</sup> using the Jena<sup>4</sup> framework. Knowledge of object-oriented programming is necessary for the understanding of the source code.

---

<sup>2</sup><http://www.w3.org/RDF/>

<sup>3</sup><http://java.sun.com/>

<sup>4</sup><http://jena.sourceforge.net/>

## Chapter 2

# Skipforward overview

As already mentioned in Section 1.1, the number of services on the internet offering information about a certain topic is overwhelming. For the area of music alone, there are many different platforms that all focus on different aspects of music. Many of these services also include webradios, where people can listen to the music immediately, and webshops to buy the discovered music.

For example, Discogs<sup>1</sup> is a “*community-built database of music information*” that provides discographies for artists and labels, detailed information about releases, and reviews.

Last.fm<sup>2</sup> on the other hand focuses on music recommendation based on the listening behavior of its users, but also offers the possibility to add user-defined tags, comments, and descriptions to artists and songs.

Pandora<sup>3</sup> also focuses on music recommendation, but does so by comparing songs through their content, which has been added by professional music-analysts describing the features of a song.

So, why would one like to extend the vast list of competing services with yet another one, dispersing the available information to more locations than they already are?

The motivation behind Skipforward is to assist the user in his search for relevant and reliable information [Kiesel and Schwarz, 2008]. He should not have to read through fifty user comments to eventually find the information he was looking for. When there are different opinions about something, the user should be provided with additional information about the reliability of statements, making it easier for him to decide on his own what to believe.

Some platform allow users to add tags to items to describe them. However, often the only way to state that a certain tag is not suitable for an item is to not add this tag. Other users can not distinguish if a tag was not added on purpose or if it was forgotten.

---

<sup>1</sup><http://www.discogs.com/>

<sup>2</sup><http://www.last.fm/>

<sup>3</sup><http://www.pandora.com/>

The system should not be limited to a certain type of features, like songs, but be easily extensible to other areas. Also, the information provided by the user should stay within his reach, as they may be personal and users may not want to feed a central server with a profile of themselves.

Thus, the requirements made for Skipforward can be summarized into the following aspects:

**Formalization** The information and opinion about items of different kinds (songs, movies, etc.) has to be formalized and stored in a machine-readable format, so that it can be used to create useful visualizations that can be easily perceived.

**Dissent** Users should be able to explicitly express their dissens with an opinion of another user. This also includes the capability to not only state that an item has a certain feature, but also that it does not have a certain feature.

**Confidence** Sometimes one is not completely sure about a feature of an item, so users should also be able to express their confidence in their own opinions. This way, others have more information to decide on the reliability of a statement.

**Decentralized storage** Not everyone wants to share his opinions with all other users for different reasons. Also, having the data of all users in one location - being controlled by a single person or organization - increases the risk of malicious usage of these personal data. Therefore, the system has to be a decentralized network.

**Easy merging of databases** With the data being stored in a decentralized manner, the effort to merge the databases of different users has to be kept low.

**Provenance** Because different information have different origins, at every moment it has to be clear where a piece of information comes from. Being able to safely identify the author of a comment or opinion also increases trust in the system.

## 2.1 Skipinions: The top level ontology

Skipforward stores the items and their features in an RDF model and formalization is realized by its toplevel ontology, called *Skipinions*, which provides the two basic classes: *Item* and *Feature*.

Every type of items that are to be annotated (e.g. books, songs, etc.) has to be a subclass of `skipinions:Item`, and instead of using different predicates for different feature types, each feature type itself is represented by a subclass of `skipinions:Feature`. These subclasses are then defined in domain ontologies, such as the *Ludopinions* ontology for board games and the *Skiptrax* ontology

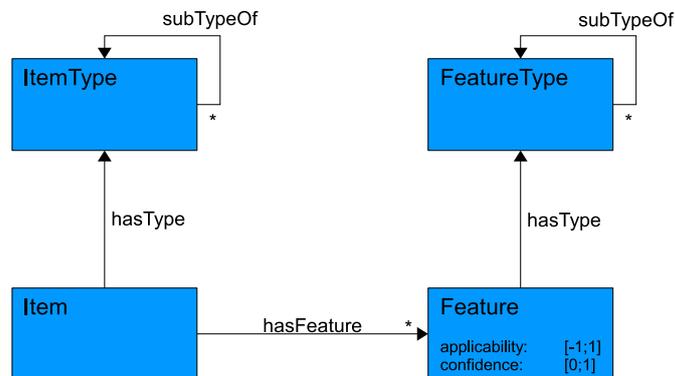


Figure 2.1: Class diagram of Items and Features

for songs (see Figure 2.1). Also, all item and feature types are required to have a title, which we will use equivalently to the actual class instead of URIs for better understanding.

To maintain consistency with the class names of the implementation, an opinion about an item will be called *feature*, whereas the *feature type* denotes the class of this feature.

The Skipinions ontology defines features to have the properties *applicability*<sup>4</sup> and *confidence*. The values for applicability reach from  $-1$  for “*doesn’t apply at all*” to  $+1$  for “*applies completely*”, where an applicability of  $0$  stands for “*can’t be decided*” or “*neither nor*”. The confidence from  $0$  for “*completely unsure*” to  $1$  for “*completely sure*”. When an opinion about an item is expressed, an instance of the appropriate feature class is created and the applicability and confidence value are added to the feature instance directly. The creation date will be added automatically by Skipforward and the property *isResponseTo* can be used to point to a previously created feature for the same item to form a discussion thread. To avoid misunderstandings, from here on we only use the term feature for instances of a feature class and feature type as a synonym for feature class.

For example, “*repetitive song structure*” is a feature type, but if Alice says that a specific song has a repetitive song structure, and that she is absolutely sure about this, Skipforward creates a new feature of the type “*repetitive song structure*” for this song item with an applicability of  $1$  and a confidence of  $1$ . Furthermore, let’s say that Bob does not agree with Alice here and has the opinion that this song does not have repetitive song structure, but isn’t quite sure about that, then the created feature will also be of the type “*repetitive song structure*”, but with an applicability of  $-1$  and a confidence of  $0.5$ .

It is important that an applicability of  $0$  and a low confidence must not be confused, as for example Carol might say that this song neither has a repetitive, nor an unrepetitive song structure (maybe because some parts are repetitive and

<sup>4</sup>also known as *truth value*

others are not), but is completely confident about this, would result in a feature with applicability 0 and confidence 1.

There are of course feature types where applicability values other than 1 or  $-1$  seem to do not make much sense, for example for the feature type “*lyrics*”, because either there are lyrics or not. Then again, some users might think otherwise, so the range of the applicability value is never restricted by the system.

So far, this definition of feature classes only allows one to state whether a specific feature is present in an item or not, which is why they are called *binary* features. Obviously, there can be only one valid feature per user, item and feature class, meaning that a second feature of the same class, from the same user and to the same item would make the old feature invalid, which reflects a change in the opinion of the user.

Additionally, subclasses of `skipinions:Feature` can have an extra slot for defining an object to the feature, like it is needed for the artist or the title of a song, and are called *non-binary* features . One basic example for a non-binary feature class is already defined in the Skipinions ontology: `skipinions:ItemName`. The *ItemName* feature class has an additional property called *itemName* that holds a string value. For example, Alice says that a song has the title “James Bond - Goldeneye (Main Theme)” and is quite sure about it, which results in a feature of type `skipinions:ItemName` with applicability 1 and confidence 0.7, and the title as *itemName* property.

Bob disagrees and on his part expresses the opinion that the song title is simply “Goldeneye” and he is very certain of it, leading to a feature with applicability and confidence 1 and the string “Goldeneye” as value for the property *itemName*. On the other hand, he thinks that the title Alice gave this song item is not completely wrong, so he also creates a feature of type `skipinions:ItemName` with the *itemName* “James Bond - Goldeneye (Main Theme)” with applicability 0.5 and confidence 1. Note, that Bob’s second opinion is a dissent with Alice’s opinion, not with his own first opinion, because the value of `skipinions:itemName` specializes the feature type.

To make the handling of applicability and confidence values more intuitive, these feature values are represented as colored circles. The bigger the circle, the higher the confidence. The color ranges from red for an applicability of  $-1$  over yellow for 0 to green for 1 (see screenshot in Figure 2.2).

## 2.2 Formal definitions

So every feature  $f$  is a tuple  $(a, c)$  where  $a$  is the applicability value and  $c$  the confidence, and since there can be at most one valid feature per user, item and feature type, we can write it as a function

$$f : U \times I \times T_f \longrightarrow [-1; +1] \times ]0; +1] \cup \{\emptyset\}$$

for binary feature types and

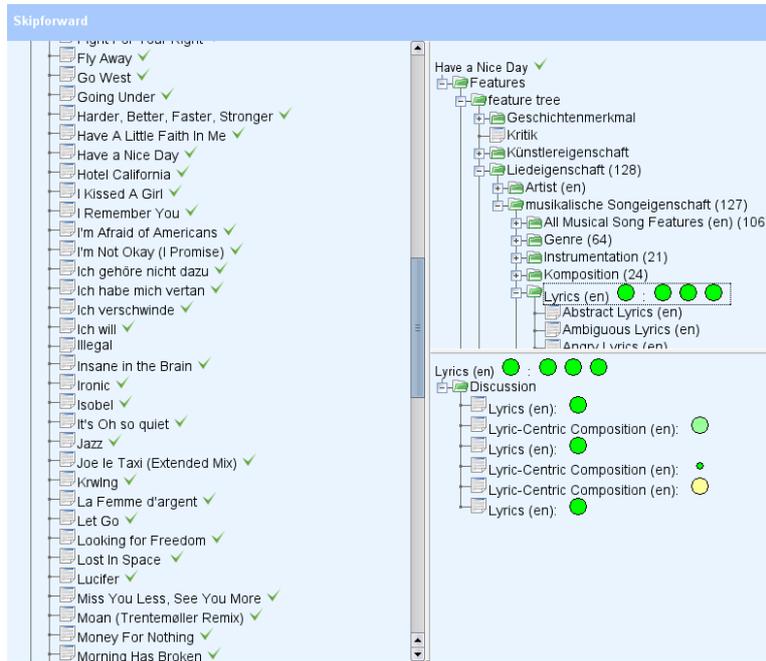


Figure 2.2: Screenshot of the web-interface of Skipforward

$$f : U \times I \times T_f \times O_{t_f} \longrightarrow [-1; +1] \times ]0; +1] \cup \{\emptyset\}$$

for non-binary feature types, where the involved sets are:

- $U$ : set of all users
- $I$ : set of all items
- $T_f$ : set of all feature types
- $O_{t_f}$ : set of allowed objects for feature type  $t_f$

and  $\emptyset$  means *undefined*. For reasons of readability and convenience, we will write a feature as variable with indices:

$$f(u, i, t_f) \stackrel{equiv}{=} f_{u,i,t_f} = (a, c)$$

$$f(u, i, t_f, o) \stackrel{equiv}{=} f_{u,i,t_f,o} = (a, c)$$

with  $i \in I, u \in U, t_f \in T_f$  and  $o \in O_{t_f}$

If it is clear from the context that the equations are applied to single feature types each, these indices will also be omitted:

$$f(u, i) \stackrel{equiv}{=} f_{u,i} = (a, c)$$

## 2.3 Maintaining information provenance and authentication

Every running Skipforward instance represents a node in the peer-to-peer network, which can be started on the user's own computer or run on a different machine that provides access through a web-interface. The user has to enter his XMPP<sup>5</sup> account information and the node then connects to the XMPP server. The nodes communicate with each other by requesting updates and sending updates via XMPP messages and file transfers.

When opinions are expressed by creating instances of feature classes, like all RDF resources they are identified using a URI. Skipforward encodes the provenance of skipinions ontology instances as the namespace that is used to create the resources: Every item or feature is created in the namespace `skip://user@host/`, where *user@host* is the users Jabber ID. Furthermore, each user only creates or modifies statements in his own namespace, which we define here as statements whose subject resource is in his own namespace.

Suppose Alice and Bob have the Jabber accounts *alice@skipforward.net* and *bob@skipforward.net* respectively, then Alice's namespace would be `skip://alice@skipforward.net/` and Bob's would be `skip://bob@skipforward.net/`. If Alice now wants to get the opinions of Bob, i.e. the feature instances he created with all their relations, she has to ask him to send him his private model. Bob will then send his private model via XMPP file transfer to Alice, who in turn will only accept statements in Bob's namespace. That way, Alice knows that all statements in Bob's namespace were received from Bob, or rather from a node that successfully authenticated with Bob's account information to the XMPP server [Wang and Vassileva, 2003]. All the *private models* of different users can be integrated into one big *world model* while one still can easily determine the provenance of statements.

The ontologies, Skipinions as well as domain ontologies, are distributed using the same mechanism. This is possible because ontologies can be in user namespaces, too. For example, *seeder@skipforward.net* is a valid XMPP account and serves as a seeder for the Skipinions ontology, which is defined in the corresponding namespace `skip://seeder@skipforward.net/`. Users can also define their own ontology in their own namespace and it will get distributed to the other nodes the same way the items and features get distributed. On the other hand, creating own accounts with the sole purpose of seeding ontologies is a good way to separate personal opinions from ontologies. This dummy users do not need to be permanently logged in into the Skipforward system, but they are the best way to propagate changes on the seeded ontologies as well as new ontologies.

---

<sup>5</sup><http://xmpp.org/>

## 2.4 Item identity

The consequence of these strictly separated namespaces is that every user can only add features to items that are in his own namespace, too.

If Alice now wants to state an opinion about one of Bob's items, there appears to be a problem, because although she would create an instance of the desired feature type in her own namespace, the RDF triple relating it to the item would have a subject that is NOT in her namespace. The solution is to use the `skipinions:isIdenticalTo` property<sup>6</sup>: Alice creates an item in her own namespace, relates it to Bob's item using this predicate and then can express opinions about this item in her own namespace (see Figure 2.3).

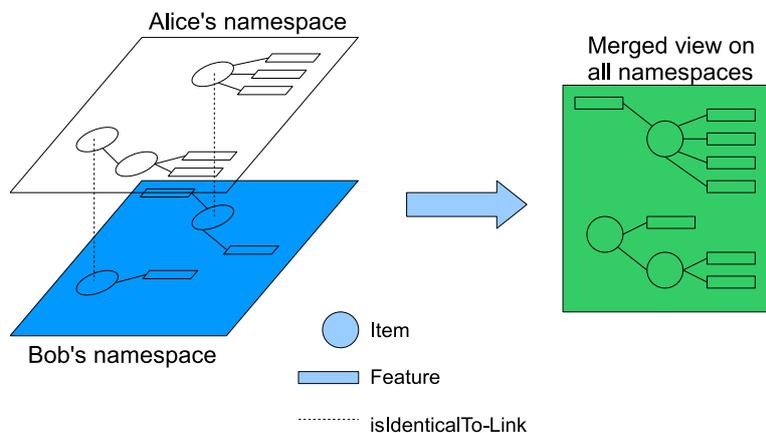


Figure 2.3: Identical items in different namespaces

When the copy of Bob's item is created in Alice's namespace, the name of the item will also be copied by creating a new `ItemName` feature in Alice's namespace with the same string for the name. This is due to two reasons:

Firstly, if another user, Carol, is a friend of Alice, but not of Bob, she only sees the items and features in Alice's and her own namespace. If Alice has not expressed an opinion about the name of that item, Carol will only see an item with no name. Secondly, if the name in Bob's namespace would get deleted for some reason, Alice would be left with an item without a name. Copying the name along with the item increases the redundancy in the network and therefore makes it more robust.

The `isIdenticalTo` relation is transitive, so if Carol copies an item from Alice that she has previously copied from Bob, it is clear that Carol means the same item as Bob.

<sup>6</sup>`skipinions:isIdenticalTo` is similar to the `owl:sameAs` property of the Web Ontology Language (OWL)

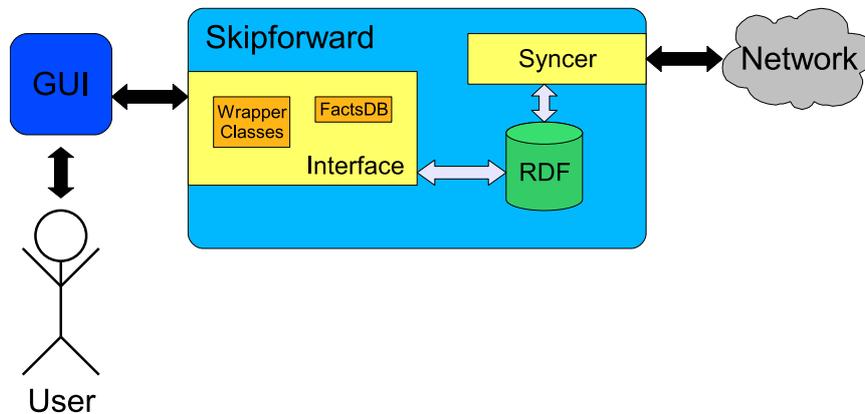


Figure 2.4: Architecture of Skipforward

## 2.5 Manual and automatic annotation

The items in the database of Skipforward are added by the users while working with the system. Without any knowledge about the features of an item it can neither contribute to the user profile nor can its usefulness be estimated, so the features have to be added by the users manually, too. This approach can obviously be impractical and heavily depends on the number and motivation of the people using the system. However, for multimedia data, such as audio and video streams, the automatic extraction of features is sometimes hard to apply, especially for features that are disputable (e.g. mood, genre, etc.). The manual annotation of items often lacks alternatives.

On the other hand, using software to automatically extract certain kinds of features and add them to the Skipforward database is not prohibited in any way and can provide useful information [Liu et al., 1998]. They will not be discussed further, though, as this thesis focuses on similarity and trust in a personalized context.

## 2.6 Architecture

Access to the data is done via the FactsDatabase component of Skipforward in an object-oriented manner, encapsulating the classes of the skipinions ontology and their instances as java classes and object respectively.

Of course, Alice and Bob do not manipulate the data on the RDF triple level. Skipforward provides an object-oriented interface for data manipulation on the abstraction level of items and features. In the above examples, Alice and Bob will be presented some graphical user interface that displays the items and their features and offers possibilities to add new ones to the database. Things like copying items to the own namespace and linking it with its source are handled outside the user's sight, and requesting and sending updates of private models happens automatically.

## Chapter 3

# Theoretical Foundations

Let us assume that Skipforward is used by many highly motivated people, who filled the database with thousands or even millions of items, and that there are useful ontologies for describing almost every aspect of an item with hundreds of feature types. Each item has been annotated by multiple users and they commented their opinions. Even under the optimistic assumption that there are no unhelpful flamewars between users, it would be unrealistic to expect all users to have the same opinion about the features of items.

When browsing through the items and viewing their features types, just displaying all features of all users or showing arithmetic means of all features would not be a personalized view and probably not very helpful, either. Information that might be useful to the user is obscured in two ways:

Firstly, the descriptions of the items may be inconsistent, making it hard to estimate their features at first glance. Secondly, the items that one considers useful will most likely represent only a little fraction of the available data, otherwise it would be sufficient to browse through the items one-by-one.

Considering the inconsistency of data, one intuitive approach is to estimate the trustworthiness of the information provided by other users. To be able to use these trust information in algorithms, a formal model needs to be defined. Section 3.1 gives an overview of different definitions and shows models of trust and their formalization.

A well-established method to filter relevant information out of large databases in a personalized way are recommendation systems. Section 3.2 gives an introduction into the two large classes of recommendation systems, namely collaborative filtering and content-based filtering.

### 3.1 Trust

The concept of trust can be applied in different situations: If we want to travel by plane, we wouldn't do so if we didn't have trust in the abilities of the pilot and the ground personal responsible for the maintenance of the aircraft. If we don't lock

our office when we fetch a cup of coffee, we do so because we trust the other people in the department to not go in there and steal our stuff. When one is confronted with contradicting statements, distinguishing the “right” information from the “wrong” is question of trust.

### 3.1.1 What is trust?

Obviously, there are different kinds of trust and not all of them are suitable for our scenario of finding relevant information. [Gutscher et al., 2008, p. 51] defines trust as a multi-relational concept:

“A *truster* trusts a *trustee* (e.g., a person, an institution or a technical system) in a certain *context*, if the truster has confidence<sup>1</sup> in the *competence* and *intention* of the trustee and therefore believes that the trustee acts and behaves in an expected way, which does not harm the truster.”

Then, trust is distinguished into two categories:

**Competence trust** Trust in the *capability* of a person, in an institution or in the functionality of a machine or a system.

**Intentional trust** Trust in the *moral integrity* (benevolence) of a person.

In the context of Skipforward, the *competence* of another user is his ability to provide correct information and his *moral integrity* means that he will not intentionally provide incorrect information. Since in this case, both the lack of competence and malevolence would lead to incorrect information, when we talk of trust we mean competence trust, where intentional trust is a special case of it. More detailed classifications of different kinds of trust can be found [Jøsang et al., 2007], but are not of interest in the context of this thesis.

### 3.1.2 Personalized views using trust

As stated at the beginning of this chapter, a simple approach of handling many different opinions about an item would be to calculate the arithmetic mean of all features of a type for an item and just use this mean value. But if one had the opinion that a song has “great lyrics” and the majority of the other users would think the opposite, this arithmetic mean wouldn’t be of much help. There is also the problem that people might have a different understanding of the meaning of a feature type, which is a problem of *ontology misunderstanding*. This is outside the scope of this thesis and we assume that all users implicitly agree on the concepts represented in the ontologies.

---

<sup>1</sup>Although the terms trust and confidence are often used synonymous, they are not exactly the same [Adams, 2005].

Assuming a user, Alice, knows that the features about “great lyrics” assigned by another user, Bob, often differ from her own opinion, then Bob’s competence of describing items with the feature type “great lyrics” would be very low relative to Alice. So Alice could decide to generally decrease the influence of Bob’s opinion about this feature type, i.e. the features of this type. At the same moment, Bob has the same reason to have little trust in Alice describing songs with this feature type, because from his point of view she is often wrong.

### 3.1.3 Trust models

It is possible to model trust values using only one scale. This scale may be discrete (e.g., “no trust”, “some trust”, “full trust”) or continuous (e.g., [0;1] as in [Maurer, 1996] or [-1;1] as in [Marsh et al., 1995]), where the exact meaning of the values differs from model to model. Most existing approaches to model trust not only allow to express the degree of trust, but also the degree of certainty or confidence in this trust statement. This can be done by adding a second scale for confidence, similar to the confidence of features in Skipforward, or by representing trust by an upper and a lower bound [Shafer, 1976].

The importance of a confidence value becomes clear when one has to assign a trust value to someone he has little information about. To be secure, one could always assign the lowest possible trust value to minimize the risk of false information, but this could also mean that valuable information are not considered because of missing trust. A common solution in this case is to express *ignorance*, which means that one lacks sufficient information and therefore has no confidence in the made trust statement.

Although continuous scales for trust and confidence values allow more detailed trust statements than discrete scales, they also make it hard to define clear semantics for them [Gutscher et al., 2008]. For example, a confidence value of 1 is twice as big as 0.5, but does not necessarily mean that the confidence is twice as big. It is also unclear if a confidence value of 0.8 means the same to different users, whereas the extreme values 0 and 1 can safely be assumed to mean the same to everyone<sup>2</sup>.

## 3.2 Recommender Systems

Recommender systems emerged in the mid-1990s and have become an important area in research and industry [Schafer et al., 1999], because their main goal is to help users deal with *information overload*. Online shops, like Amazon.com<sup>3</sup>, use collaborative filtering techniques to recommend products to the customer that he might find interesting, based on the products other customers bought previously [Linden et al., 2003], other platforms like Last.fm<sup>4</sup> recommend artists and songs

---

<sup>2</sup>Of course, the same holds for the trust values.

<sup>3</sup><http://www.amazon.com/>

<sup>4</sup><http://www.last.fm>

based on the listening behavior of their users.

The *recommendation problem* can be commonly described as the problem of estimating the *utility* of an item to a user, that he has not seen or rated, yet. The items with the highest estimated *usefulness* will then be presented to him. This estimation is based on *ratings* previously given by this user to other items and further information, depending on the type of the recommendation system. Often the user can assign the ratings manually, but it can also be implicitly gathered, e.g., by monitoring his shopping or listening behavior.

It should be noted that the term rating is often used to describe how the user rates the utility of an item. Because in most recommender systems user can only give an overall rating, *rating* and *utility* are often used synonymously.

Formally, the estimated usefulness can be seen as a function

$$pred : U \times I \longrightarrow R \quad (3.1)$$

where  $U$  is the set of all user,  $I$  the set of all items and  $R$  is a totally ordered set. As mentioned earlier, different recommender systems have different methods of predicting the usefulness and use different data for it.

### 3.2.1 Collaborative Filtering

Collaborative filtering (CF) systems first calculate the similarity between all pairs of users based on the co-rated items, i.e., items that have been rated by both users, and denotes the highly similar users as *peers* or *neighbors*. The highest rated items of these neighbors that are not already rated by the active user, i.e., the user currently interacting with the system, are then recommended to him. The assumption behind this behavior is that users who rated the same items similarly have similar taste and, hence, the rating of new item will be similar to those of similar users [Sarwar et al., 2000].

Since the recommendations are based on the similarity of users, this method is called *user-based* CF. Another method is to calculate the similarity between items and then recommend items that are similar to the ones already rated highly. Two items are considered similar, if they received similar ratings from the same users (e.g., *users who like item A also like item B* and *users who don't like item A also don't like item B*). A popular example for *item-based* CF are the recommendations from Amazon.com: "*Users who bought A also bought B.*"

It should be noted that the only information used about the items is their ratings from different users, no information about their content is needed. The ratings can be stored in a so-called user-item matrix, where rows are users and columns are items (see Figure 3.1). A column now represents all ratings by all users for this particular item, whereas rows represent all ratings of all items of a specific user. Calculating the similarity of users or items can thus be seen as a comparison between rows or columns respectively, which both can be interpreted as vectors.

Additionally, CF algorithms can be divided into two general classes, *memory-based* and *model-based*. As the name suggests, model-based CF tries to learn a

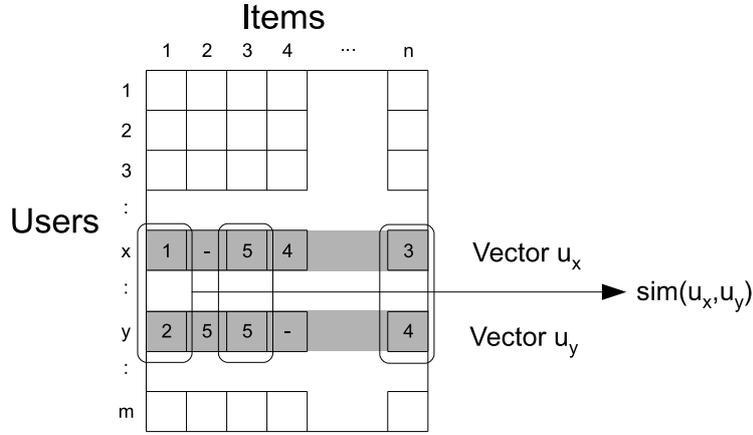


Figure 3.1: User-Item Matrix where two users are compared.

*model* of the user from the collected ratings and calculates its predictions based on this model. Memory-based CF utilizes the entire user-item matrix and uses heuristics to calculate similarities. In [Sarwar et al., 2001], the classes memory-based and user-based are assumed equal, same with model-based and item-based, but there are also model-based methods that are user-based [Adomavicius and Tuzhilin, 2005].

### The Pearson correlation

A common measure used in CF is the *Pearson product-moment correlation coefficient*, or *Pearson correlation coefficient* for short, as it measures the linear relationship between two random variables  $X$  and  $Y$ . It can be interpreted as the normalized covariance of both random variables [Rodgers and Nicewander, 1988] and is of range  $[-1; +1]$ :

$$\begin{aligned}
 \rho_{X,Y} &= \frac{cov(X, Y)}{\sqrt{var(X)var(Y)}} \\
 &= \frac{\mathbb{E}[(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))]}{\sqrt{(X - \mathbb{E}(X))^2(Y - \mathbb{E}(Y))^2}}
 \end{aligned}
 \tag{3.2}$$

If it is calculated from a sample of these random variables, it is written as:

$$r_{X,Y} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}
 \tag{3.3}$$

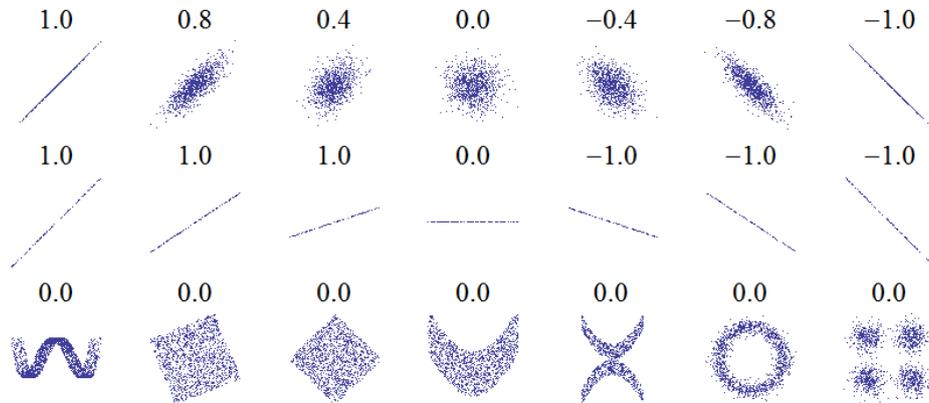


Figure 3.2: Correlation examples

(Source: <http://en.wikipedia.org/wiki/Correlation>, Dec 14th, 2008)

where  $\bar{x}$  and  $\bar{y}$  are the sample means of  $X$  and  $Y$ .

In CF, the rating behavior of a user can be viewed as a random variable, and the ratings already assigned represent a sample of this random variable. Given two users  $u_x$  and  $u_y$ , the Pearson correlation coefficient is calculated over all co-rated items, i.e., items both users have rated, denoted by  $I_{xy}$ , whereas  $r_{x,i}$  is the rating of user  $u_x$  for item  $i$  and  $\bar{r}_x$  is his average rating for *all* items.

$$\text{sim}(u_x, u_y) = \frac{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r}_x)(r_{y,i} - \bar{r}_y)}{\sqrt{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r}_x)^2 \sum_{i \in I_{xy}} (r_{y,i} - \bar{r}_y)^2}} \quad (3.4)$$

As stated above, the correlation coefficient takes values from  $-1$  to  $1$ . A value of  $1$  indicates a perfectly linear relationship between both random variables, meaning that when plotted as data points  $(x_i, y_i)$ , all those points lie on the same line and this line has positive slope, whereas a value of  $-1$  means that all points lie on a line with negative slope.

If both variables are independently distributed the correlation coefficient is  $0$ , but the inversion is not true. The correlation coefficient can be  $0$  even if both variables are fully related in some non-linear way. Some examples are shown in Figure 3.2: The first row shows plots of variables with different degrees of correlation, the second row shows data sets with a linear dependency, and the last row shows variables that have non-linear relationships. Note, that the example in the middle has a correlation of  $0$ , because the variance of  $Y$  is  $0$ .

In general, the Pearson correlation coefficient is not sufficient to evaluate the relationship between two random variables and other measures may be more useful [Carroll, 1961]. A prominent example that shows that the correlation coefficient has to be handled with care was given by [Anscombe, 1973].

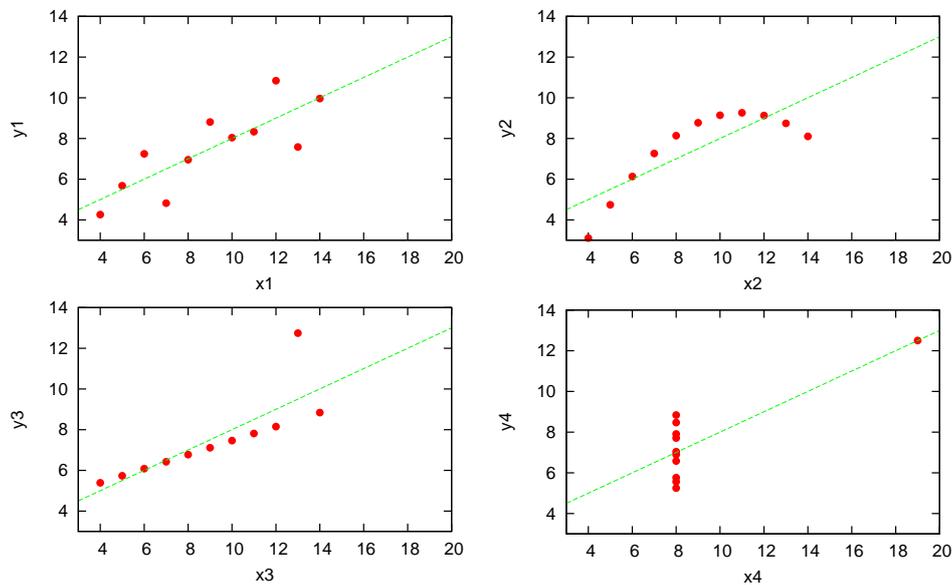


Figure 3.3: The plots of Anscombe’s Quartet. The green line is the regression line, which is the linear function with the least square error for the data.

Figure 3.3 shows the plots of *Anscombe’s quartet*, consisting of four data sets with two variables  $X$  and  $Y$  that all have the same simple statistical properties, such as mean, variance and regression line. In all four examples, both random variables have a correlation of 0.816, but the plot shows that the dependencies between these variables are very different.

However, the Pearson correlation has shown to produce useful results for in collaborative filtering, so it is a good candidate for calculating similarities between users and their opinions.

### 3.2.2 Content-based Filtering

Content-based filtering (CBF) systems estimate the usefulness of an item unknown to the user by comparing it to the items that this user has already rated. So far, this sounds like the item-based CF, but here the items are compared by their *content* and not their rating from other users. The content of an item can generally be described through attributes or features of different classes (e.g., binary, nominal, ordinal, etc.) and is called an *item profile*. In the same way, the information about the users preferences are called *user profile* and the user can specify directly what contents he likes or dislikes (“I like Jazz, but I don’t like male vocals”) or indirectly by denoting items he likes or dislikes (“I like songs A and B, but I don’t like song C”) and the system tries to derive the contents liked and disliked from this set.

The implicit construction of a user profile through rated items can again be

divided into the two classes *memory-based* (or *heuristic-based*), where rated items are directly compared to the unrated items, and *model-based* techniques, that learn a model from the rated items that will be used to predict the usefulness of unrated items, such as artificial neuronal networks or decision trees.

### **3.2.3 Hybrids**

Although each of the approaches described above can be applied on their own, there are many kinds of recommendation systems that use some kind of combination of both filtering methods or other approaches not listed here [Balabanović and Shoham, 1997]. The aim is to compensate disadvantages of one kind of systems by integrating techniques of the other one.

For example, if there are not enough co-rated items to calculate user similarities in a collaborative system, content profiles can be used to find similar users, thus allowing to overcome the sparsity problem [Pazzani, 1999].

Keeping this in mind, when we use the terms CF or CBF, we refer to the “pure” idea without any combination or augmentation with other methods.

## Chapter 4

# Approach

The basic idea of this thesis is to create a personalized view of the inconsistent data by automatically calculating a similarity between pairs of users, based on the existing annotations stored in the database. The user similarity is then transformed into a competence trust value to determine the influence another user's annotations should have on the aggregated view of the item [Massa and Avesani, 2004].

In Section 4.1 we will first examine how collaborative filtering systems calculate their user similarity values and then derive an algorithm that works with the data provided by Skipforward with respect to computational performance. Based on this results, in Section 4.2.2 a trust model is developed to represent the competence trust in other users that is used to create a personalized view on data [Ziegler and Lausen, 2004].

### 4.1 User similarity

Because user-based CF also calculates the similarity between users based on the existing data, it is reasonable to assume that CF algorithms provide a good foundation for calculating user similarities for content-based data. We decided to concentrate on the Pearson correlation, as it is widely applied in collaborative recommendation systems and it is known to yield good results.

As stated earlier, the difference between content-based and collaborative filtering is that CF only considers one value per user and item, the rating, whereas in content-based filtering there are multiple values per item and these values can also be of different type. Skipforward can be seen as a hybrid system, because there are multiple values per item like in CBF, but they are contributed by the users and may be subjective and inconsistent like in CF (see Table 4.1).

The idea here is to apply the Pearson correlation on one feature type at a time, which will result in a correlation coefficient that describes how similar two users are regarding this specific feature type. To reduce complexity, we will ignore non-binary feature types in this diploma thesis.

Table 4.1: Differences between features in CF, CBF and Skipforward

	CF	CBF	Skipforward
<b>Number</b>	1	many	many
<b>Types</b>	rating	content	anything
<b>Subjective</b>	yes	no	yes

#### 4.1.1 Adapting the Pearson correlation

Normally, the Pearson correlation is applied to the one-dimensional rating values  $r$ , whose equivalent value in Skipforward is the applicability value  $a$  of a feature<sup>1</sup>. But the features in Skipforward have an additional confidence value  $c$ , which is not considered in the standard Pearson correlation. To solve this shortcoming, we have to think about what effect the confidence value has on a feature's meaning and therefore on the similarity between users that has to be defined yet.

When two users have directly opposite opinions about a certain aspect of an item (i.e., one applicability is  $-1$  and the other one is  $+1$ ), they can have different confidence in their opinions. If they are both very certain of their opinion, it should have a more negative impact on the user similarity than when they both are unsure. Thus, a reasonable interpretation of the confidence value in this context would be a weighting.

There exists a weighted Pearson product-momentum correlation coefficient, which is defined as

$$r_{X,Y} = \frac{\sum_{i=1}^n w_i (x_i - \bar{x}_w)(y_i - \bar{y}_w)}{\sqrt{\sum_{i=1}^n w_i (x_i - \bar{x}_w)^2 \sum_{i=1}^n w_i (y_i - \bar{y}_w)^2}} \quad (4.1)$$

Here,  $\bar{x}_w$  and  $\bar{y}_w$  denote the weighted means, defined as

$$\bar{x}_w = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i} \quad \text{and} \quad \bar{y}_w = \frac{\sum_{i=1}^n w_i y_i}{\sum_{i=1}^n w_i}$$

This leads to the modified version of equation 3.4 of the user similarity

---

<sup>1</sup>As already mentioned in Section 3.2, the term rating mostly describes how much a user likes an item or how much he considers it useful. The term *applicability*, however, only makes sense in the context of a specific feature type. In the following, we will stick with the term *rating* and the variable name  $r$  to be compatible with the literature on collaborative filtering.

$$sim(u_x, u_y) = \frac{\sum_{i \in I_{xy}} w_i (r_{x,i} - \bar{r}_x)(r_{y,i} - \bar{r}_y)}{\sqrt{\sum_{i \in I_{xy}} w_i (r_{x,i} - \bar{r}_x)^2 \sum_{i \in I_{xy}} w_i (r_{y,i} - \bar{r}_y)^2}} \quad (4.2)$$

with

$I$ :	set of all items
$I_x = \{i \in I   r_{x,i} \neq \emptyset\}$ :	set of all items rated by user $u_x$
$I_y = \{i \in I   r_{y,i} \neq \emptyset\}$ :	set of all items rated by user $u_y$
$I_{xy} = I_x \cap I_y$ :	set of all items rated by user $u_x$ and $u_y$

But this equation still doesn't fit our purpose, because it allows only one weight value per rating value to be defined, meaning that the ratings of both users have to be weighted with the same weight value  $w_i$ . It might seem possible to have two different weight values in the denominator, a weight  $w_{x,i}$  for the variance of user  $u_x$  and a weight  $w_{y,i}$  for the variance of user  $u_y$ , but the numerator only contains a single weight value for both the rating value of user  $u_x$  and  $u_y$ .

We looked into the possibility to use two different weight values  $w_{x,i}$  and  $w_{y,i}$  in the denominator and a combined weight value  $w_{xy,i}$  in the numerator. If the combined weight fulfills some criteria, it is possible to prove that the resulting correlation coefficient still has a range of  $[-1; +1]$ , but it would be necessary to examine if the resulting equation still has the properties of a correlation. If only one combined weight value  $w_{xy,i}$  is used in the equation, it is the normal weighted Pearson correlation and we do not need to worry about its properties. We decided to consider both variants in our algorithm and compare their results based on evaluation data.

In both cases, we need to find a function to calculate a combined weight value out of the weight values from both users. The choice of this function is not arbitrary, as the combined weight has to reasonably represent the combination of two confidences. Desired properties are:

- The confidence value is of *range*  $[0; +1]$ , so the combined weight/confidence should be, too.
- If one of the users is completely unsure of his opinion (i.e., a confidence of 0), the similarity of the corresponding rating should not be considered (i.e., a combined weight value of 0).
- If both users have the same amount of confidence in their opinion (i.e., the same confidence value), the combined weight value should be the same.

Two candidates for calculating the combined weight that fulfill these criteria are

$$w_{xy,i} := \min(c_{x,i}, c_{y,i}) \quad (4.3)$$

$$w_{xy,i} := \sqrt{c_{x,i}c_{y,i}} \quad (4.4)$$

where (4.3) is the minimum of both weights and (4.4) their geometrical mean. The weight values  $w_{x,i}$  and  $w_{y,i}$  are defined as

$$\begin{aligned} w_{x,i} &:= c_{x,i} \\ w_{y,i} &:= c_{y,i} \end{aligned} \quad (4.5)$$

if we want to use user-specific weight values in the denominator and

$$w_{x,i} = w_{y,i} := w_{xy,i} \quad (4.6)$$

if we use only the combined weight value for the weighted Pearson correlation, which will then look like this:

$$\text{sim}(u_x, u_y) = \frac{\sum_{i \in I_{xy}} w_{xy,i} (r_{x,i} - \bar{r}_x)(r_{y,i} - \bar{r}_y)}{\sqrt{\sum_{i \in I_{xy}} w_{x,i} (r_{x,i} - \bar{r}_x)^2 \sum_{i \in I_{xy}} w_{y,i} (r_{y,i} - \bar{r}_y)^2}} \quad (4.7)$$

with the weighted mean defined as

$$\bar{r}_x = \frac{\sum_{i \in I_x} w_{x,i} r_{x,i}}{\sum_{i \in I_x} w_{x,i}} \quad \text{and} \quad \bar{r}_y = \frac{\sum_{i \in I_y} w_{y,i} r_{y,i}}{\sum_{i \in I_y} w_{y,i}} \quad (4.8)$$

#### 4.1.2 Limits of the Pearson correlation

An often instanced advantage of the Pearson correlation is that it produces useful results even if the two compared sample sequences have different ranges. In collaborative filtering systems, the range of rating values out of which users can choose is usually the same for every user, e.g., a discrete scale from 1 to 10, but the users may still use only parts of this range as ratings. Alice might only rate items with values from 3 to 10, whereas Bob rates items from 1 to 8. For the Pearson correlation it makes no difference if the two users could not use other values or if they just did not use them, which leads to a behavior that is desirable in the context of CF. As an example, imagine Alice rated item A with 5 and item B with 10, and Bob rated the same items with 1 and 3, then the Pearson correlation coefficient for those two users is exactly 1. Why is that?

The Pearson correlation centers the rating values to 0 when calculating the variance and covariance. The mean rating of Alice is 2, the mean rating of Bob is 7.5, which means that for both items both users differ from their mean rating

into the same direction, i.e., a rating less than the mean or a rating greater than the mean. Both users rated item B higher than item A, which leads to a correlation of +1, because the Pearson correlation indicates the strength of a linear relationship (see Chapter 3.2.1). Between only two points, a connecting line can always be plotted, so unless one of the two variables (i.e., the ratings of one user) has no variance because both values are the same, the correlation will always be +1 or -1.

If the strategy of a collaborative filtering algorithm is to recommend the highest ranked items of the neighbors (i.e., the users with the highest similarity) then it does not matter if the other user's ratings have a different range, because only the relative ordering of his items is important. In Skipforward, however, we don't want to get the item with the highest applicability for a certain feature type, but to predict the personal opinion about the applicability of a feature type for a specific item based on the features other users assigned to it.

The sparsity of data is common problem for recommendation systems [O'Donovan and Smyth, 2005], [Massa and Bhattacharjee, 2004], especially for new users, because the system does not have enough data to compare the other users with the new one. In conjunction with the usage of the Pearson correlation, this problem can increase in Skipforward because of the lack of variance in the applicability of some feature types. If, for example, the first couple of songs the new user has annotated do not have a flute solo, all features of type "flute solo" he added will most likely have an applicability value of -1, which means they have no variance because they are equal to the mean value of -1. But the normal Pearson correlation requires variance in at least one of the two random variables if the correlation is to be defined and non-zero (see equation (4.2)). So although most other users may agree with him in the applicability of the feature type "flute solo" in all of the annotated songs, the calculated correlation will be not defined or zero.

With an increasing amount of annotations made, the probability of having variance in all feature types increases and the problem vanishes. But users often judge things by their first impression, so we have to find a way to provide useful predictions as soon as possible.

### 4.1.3 The constrained Pearson correlation

In [Shardanand and Maes, 1995], a variant of the Pearson correlation is introduced that takes the *positivity* and *negativity* of ratings into account, which they called "Constrained Pearson r Algorithm". They used a discrete scale of rating values from 1 to 7 in their recommender system "Ringo", where values greater than 4 represent a positive and values below 4 a negative rating. The desired behavior of the constrained Pearson correlation is that it only increases if the two users rated an item both positive or both negative, so instead of subtracting the mean from each rating, the neutral value 4 is subtracted.

The neutral value of the applicability of features in Skipforward is 0, so we substitute the  $(r_i - \bar{r})$  terms with  $(r_i - 0)$  or simply  $r_i$ :

$i$	$r_{x,i}$	$r_{y,i}$	normal	constrained
1	1.0	1.0	0.00	1.00
2	0.0	1.0	0.00	0.71
3	-1.0	0.0	0.87	0.50
4	-1.0	-1.0	0.82	0.67

Table 4.2: Example results for both correlation variants. The weights for all ratings are set to 1 here for simplification.

$$\begin{aligned}
sim(u_x, u_y) &= \frac{\sum_{i \in I_{xy}} w_{xy,i} (r_{x,i} - 0)(r_{y,i} - 0)}{\sqrt{\sum_{i \in I_{xy}} w_{x,i} (r_{x,i} - 0)^2 \sum_{i \in I_{xy}} w_{y,i} (r_{y,i} - 0)^2}} \\
&= \frac{\sum_{i \in I_{xy}} w_{xy,i} r_{x,i} r_{y,i}}{\sqrt{\sum_{i \in I_{xy}} w_{x,i} (r_{x,i})^2 \sum_{i \in I_{xy}} w_{y,i} (r_{y,i})^2}} \tag{4.9}
\end{aligned}$$

Another advantage of the constrained algorithm is that one commonly rated item is sufficient for a correlation that is non-zero (if the ratings are non-zero, of course), which means that user similarities can be detected earlier than with the standard Pearson correlation, which needs at least two commonly rated items.

An example calculation can be seen in Table 4.2: In each line the correlations are calculated using only the ratings of this item and previous ones (e.g., the correlation in line 2 only include the ratings for item 1 and 2). The normal Pearson correlation coefficient is undefined (and therefore set to 0) in the first line, because it needs at least two ratings to be defined, and in the second line, because there is no variance in the ratings of user  $u_y$ . However, the constrained Pearson correlation already shows reasonable result with only one item being co-rated.

An evaluation conducted with “Ringo” showed that in its case the constrained Pearson r algorithm performed best, both in that it was able to provide predictions in more cases and that these predictions were more accurate than the normal Pearson r algorithm [Shardanand and Maes, 1995]. However, since we intend to use these algorithms in a very different scenario, the assumption that these results hold for Skipforward, too, has to be proven in an evaluation.

#### 4.1.4 Computational complexity and incremental calculation

All of the above mentioned variants of the Pearson correlation can be implemented straight forward: For each user  $u_x$ , the correlation with each of the other users has to be calculated based on the features of items they both have annotated.

**Algorithm 4.1.1:** CALCULATEALLCORRELATIONS( $t_f$ )

**for each**  $u_x \in U$   
**for each**  $u_y \in U$   
**do** *CalculateCorrelation*( $u_x, u_y, t_f$ )

If there are  $m$  users and  $n$  items, then there are  $\frac{m(m-1)}{2}$  correlations to be calculated and each calculation has to examine at most  $n$  items, so this naive approach has a computational complexity of  $O(m^2n)$ . The correlation values will be valid as long as no new ratings are submitted or existing ones are updated. In the event of a change in the ratings of user  $u_x$ , his correlation coefficients to each of the other  $m - 1$  users has to be recalculated, which has a complexity of  $O(mn)$ .

In collaborative recommender systems, user similarities are used to find good recommendations based on similar users and usually do not change significantly in a short time. So instead of recalculating the correlation coefficients every time a user asks for a recommendation, many systems with a large number of user and items often recalculate all user similarities once in a while, because the impact of few changes onto the recommendation process itself is often too negligible to justify a permanently updated user similarity matrix [Adomavicius and Tuzhilin, 2005].

Skipforward has a bit different requirements, because the purpose of user similarities is not only to help the system make recommendations, but to provide a personalized view while browsing through the items. Additionally, Skipforward is a decentralized system where every user has his own node in the network<sup>2</sup> and communicates only with the nodes in his roster (his *friends*). As a consequence, every node has to calculate the user similarities himself, but only needs to do so for the similarity between the own user and each of his friends, not between the friends. So if a user has  $m$  friends, the costs for computing the correlation coefficients for all of his friends are in  $O(mn)$ .

An approach to reduce the computational costs of maintaining the user similarities is presented in [Papagelis et al., 2005], by updating the similarity values if a rating has changed instead of recalculating them from scratch. For example, if there are  $n'$  rated items and a new weighted rating  $r_{x,a}$  for item  $a$  is submitted, the update of the weighted mean of ratings of user  $u_x$  would look like this:

$$\bar{r}'_x = \frac{\sum_{i=1}^{n'+1} w_{x,i} r_{x,i}}{\sum_{i=1}^{n'+1} w_{x,i}} = \frac{\sum_{i=1}^{n'} w_{x,i} r_{x,i} + w_{x,a} r_{x,a}}{\sum_{i=1}^{n'} w_{x,i} + w_{x,a}} = \frac{\left( \sum_{i=1}^{n'} w_{x,i} \right) \bar{r}_x + w_{x,a} r_{x,a}}{\sum_{i=1}^{n'} w_{x,i} + w_{x,a}} \quad (4.10)$$

<sup>2</sup>Even if multiple Skipforward nodes run on the same machine or even in the same Java VM, they are still separated instances and communicate only through XMPP messages and filetransfers.

As this example shows, instead of iterating over all  $n$  items with a complexity of  $O(n)$ , the new weighted mean  $\bar{r}'_x$  can be calculated by updating the old weighted mean  $\bar{r}_x$  in constant time  $O(1)$ , if  $\bar{r}_x$  and the old sum of weights  $\sum_{i=1}^{n'} w_{x,i}$  are cached. In the same way, the correlation coefficients can be updated with a constant effort per changed rating, leading to a complexity of  $O(m)$  for updating the user similarities to  $m - 1$  other users.

Unfortunately, the equation for calculating the correlation is more complex than the equation for the weighted mean, which makes it even more difficult to define an incremental update for it. The basic idea of [Papagelis et al., 2005] to simplify this problem is to split equation (3.4) into three factors  $B$ ,  $C$ , and  $D$ :

$$\text{sim}(u_x, u_y) = A = \frac{B}{\sqrt{C}\sqrt{D}} = \frac{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r}_x)(r_{y,i} - \bar{r}_y)}{\sqrt{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r}_x)^2} \sqrt{\sum_{i \in I_{xy}} (r_{y,i} - \bar{r}_y)^2}} \quad (4.11)$$

Instead of updating the correlation with a single rule, increments  $e$ ,  $f$  and  $g$  are determined to update each of this factors independently. The new correlation  $A'$  is then easily computed using the new factors  $B'$ ,  $C'$  and  $D'$ :

$$A' = \frac{B'}{\sqrt{C'}\sqrt{D'}} = \frac{B + e}{\sqrt{C + f}\sqrt{D + g}} \quad (4.12)$$

Since we want to implement different variants of the Pearson correlation and use weighted rating, we can only use this basic idea and have to deduce our own update rules.

For reasons of better readability, we will differ from the notation used so far. The following variables are used:

$x_i, y_i$	ratings of users $u_x$ and $u_y$ for item $i$ , respectively
$w_{x,i}, w_{y,i}$	weights of the ratings of users $u_x$ and $u_y$ for item $i$ , respectively
$w_{xy,i}$	combined weight of both users $u_x$ and $u_y$
$\bar{x}, \bar{y}$	weighted mean of all ratings of users $u_x$ and $u_y$ , respectively

The Pearson correlation from equation (4.1) is therefore split into the three factors

$$\begin{aligned} B &= \sum_{i \in I_{xy}} w_{xy,i} (x_i - \bar{x})(y_i - \bar{y}) \\ C &= \sum_{i \in I_{xy}} w_{x,i} (x_i - \bar{x})^2 \\ D &= \sum_{i \in I_{xy}} w_{y,i} (y_i - \bar{y})^2 \end{aligned} \quad (4.13)$$

The *active item* is the item whose rating has changed. Furthermore, primed variables denote the new value after the update to distinguish them from the old values:

$$\begin{array}{ll} x_a, w_{x,a} & \text{the old rating and its weight of the active item} \\ x'_a, w'_{x,a} & \text{the new rating and its weight of the active item} \\ \bar{x}' & \text{new weighted mean of all ratings of user } u_x \end{array}$$

The difference between new and old values is defined as

$$dx_a = x'_a - x_a \quad \Leftrightarrow \quad x'_a = x_a + dx_a$$

and analogously for  $d\bar{x}$  and  $dw_a$ .

There are three different kinds of changes in ratings: A new rating can be added and an existing one can be modified or deleted. Since every rating is weighted, adding a new rating or deleting an old rating can be interpreted as a change of an existing rating by modifying its weight from zero to non-zero or vice versa.

[Papagelis et al., 2005] also distinguish between the two cases of the other user having also rated the active item or not. Because the sums in the equation only accumulate ratings of co-rated items, in the latter case only mean rating of the active user changes, requiring less computations than if the set of co-rated items changes.

However, since we use weighted ratings and these weights may be defined as a combination of the confidences of both users, the change of a rating of user  $u_x$  can also effect the weighted mean of user  $u_y$ . Thus, we only need to consider one case and therefore need to deduce only one set of update rules<sup>3</sup> for  $B$ ,  $C$  and  $D$ :

$$\begin{aligned} B' = B &+ dw_{xy,a}(x'_a - \bar{x}')(y_a - \bar{y}) + w_{xy,a}dx_a(y_a - \bar{y}) \\ &- d\bar{x}\left(\sum_{i \in I_{xy}} w_{xy,i}y_i - \bar{y} \sum_{i \in I_{xy}} w_{xy,i}\right) \end{aligned} \quad (4.14)$$

$$\begin{aligned} C' = C &+ dw_{x,a}(x'_a - \bar{x}')^2 + 2w_{x,a}dx_a(x_a - \bar{x}') + w_{x,a}dx_a^2 \\ &- 2d\bar{x}\left(\sum_{i \in I_{xy}} w_{x,i} - \bar{x} \sum_{i \in I_{xy}} w_{x,i}\right) + d\bar{x}^2 \sum_{i \in I_{xy}} w_{x,i} \end{aligned} \quad (4.15)$$

$$D' = D + dw_{y,a}(y_a - \bar{y})^2 \quad (4.16)$$

At first glance, this update rules can not be calculated in constant time, because they contain factors that need to be summed over items. The solution is to cache that elements of the equation so that no iterations over items has to be made. These

<sup>3</sup>The derivation of this update rules is given in Appendix B.

elements have to be updated, too, but this is a rather trivial task and will not be explained further.

Values that need to be cached for this updates are:

- $B, C, D$   
the factors to calculate  $A'$  (for each pair of users)
- $\bar{x}, \bar{y}$   
the weighted means of all ratings (for each user)
- $\sum_{i \in I_x} w_{x,i}, \sum_{i \in I_y} w_{y,i}$   
the total weight of all ratings (for each user)
- $\sum_{i \in I_{xy}} w_{xy,i} x_i, \sum_{i \in I_{xy}} w_{xy,i} y_i$   
the  $w_{xy,i}$ -weighted sum of ratings for co-rated items (for each pair of users)
- $\sum_{i \in I_{xy}} w_{x,i} x_i, \sum_{i \in I_{xy}} w_{y,i} y_i$   
the  $w_{x,i}/w_{y,i}$ -weighted sum of ratings for co-rated items (for each pair of users)
- $\sum_{i \in I_{xy}} w_{xy,i}$   
the total combined weight for all co-rated items (for each pair of users)
- $\sum_{i \in I_{xy}} w_{x,i}, \sum_{i \in I_{xy}} w_{y,i}$   
the total weight for all co-rated items (for each pair of users)

The mean can also be updated incrementally, but we can generalize equation (4.10), which can only be used if a new rating is added, to a form that updates the mean if an existing rating is modified.

$$\bar{r}'_x = \frac{\left( \sum_{i \in I_x} w_{x,i} \right) \bar{x} - w_{x,a} x_a + w'_{x,a} x'_a}{\sum_{i \in I_x} w_{x,i} - w_{x,a} + w'_{x,a}} \quad (4.17)$$

If a rating is added or deleted this equation can also be used with  $w'_{x,a}$  or  $w_{x,a}$  set to 0.

## 4.2 Creating personalized views

In collaborative filtering systems, user similarities can easily be used to make *rating predictions* for items that the user has not rated yet [Adomavicius and Tuzhilin, 2005]. When a user browses items in Skipforward, the personalized view should

not only include a prediction of his opinion about a feature type that he has not expressed yet, but should also aggregate the features assigned by the other users for feature types he already stated an opinion. We therefore use the more general term *feature aggregation* and view the *feature prediction* function as a special use case of it.

Often, only the ratings of the most similar users are included, which is denoted by  $\hat{U}$ . Some examples are:

$$r_{x,i} = \frac{1}{|\hat{U}|} \sum_{u_y \in \hat{U}} r_{y,i} \quad \text{simple mean} \quad (4.18a)$$

$$r_{x,i} = k \sum_{u_y \in \hat{U}} \text{sim}(u_x, u_y) r_{y,i} \quad \text{weighted mean} \quad (4.18b)$$

$$r_{x,i} = \bar{r}_x + k \sum_{u_y \in \hat{U}} \text{sim}(u_x, u_y) (r_{y,i} - \bar{r}_y) \quad \text{adjusted weighted mean} \quad (4.18c)$$

$$\text{where } k = \frac{1}{\sum_{u_y \in \hat{U}} \text{sim}(u_x, u_y)} \quad \text{normalization factor}$$

The simple mean in (4.18a) is a naive approach and can only be considered as “personalized” since only the ratings of very similar users are included. The weighted mean in (4.18b) is a bit more complex as the influence of the a users rating increases with his similarity, the adjusted weighted mean (4.18c) also takes into account the possibly different use of the ratings scale as described in Section 4.1.2.

Due to the fact that Skipforward is not a centralized system, where the data of all users is stored, each decentralized node only has access to a limited amount of information determined by the users in the roster. Because of this, a further restriction of the features used for the personalized aggregation seems not sensible, as the features of users with a low similarity get a low weighting anyway.

#### 4.2.1 Confidence in similarities

The user similarities calculated by our algorithm could be used directly in conjunction with an adapted form of equation (4.18), but this would be equal to a trust model with only one scale, whose disadvantages are described in Section 3.1.3. Without the possibility of describing the confidence in this user similarity the system would be prone to attacks, because as shown in Section 4.1.2 the Pearson correlation between two users will be +1 if they share the same ratings on exactly two items, and they only need to share a rating on one item if the constrained Pearson correlation is used. This behavior of the correlation coefficient can easily be exploited by an attacker to manipulate the estimated rating (*push/nuke attack*) of an item unrated by the attacked user [O’Mahony et al., 2002].

A user similarity based on hundred items should weigh more than one only based on two ratings, so the confidence in the user similarity should therefore be based on the amount of underlying data that led to the similarity value. Possible measures are:

$$conf_{sim_1}(u_x, u_y) = \frac{|I_{xy}|}{|I|} \quad (4.19a)$$

$$conf_{sim_2}(u_x, u_y) = \frac{2|I_{xy}|}{|I_x| + |I_y|} \quad (4.19b)$$

$$conf_{sim_3}(u_x, u_y) = \frac{|I_{xy}|}{|I_x \cup I_y|} \quad (4.19c)$$

The ratio of common rated items to all items in (4.19a) depends on the total number of items in the database and leads to mostly low confidence values. (4.19b) and (4.19c) depend on the number of items that have been rated by at least one of the two users. Some example values are shown in Table 4.3.

Again, if the confidence values in Skipforward are interpreted as weights, the equations can be modified to make certain features count more than uncertain ones:

$$conf_{sim}(u_x, u_y) = \frac{\sum_{i \in I_{xy}} w_{xy,i}}{|I|} \quad (4.20a)$$

$$conf_{sim}(u_x, u_y) = \frac{2 \sum_{i \in I_{xy}} w_{xy,i}}{\sum_{i \in I_x} w_{x,i} + \sum_{i \in I_y} w_{y,i}} \quad (4.20b)$$

Equation (4.19c) can not be adapted that easily, because it is hard to define what weights have to be used for the items of  $|I_x \cup I_y|$ . Since the weights are of range  $[0; +1]$ , it is clear that in (4.20a) the confidence  $conf_{sim} \in [0; +1]$ , too, and the same can be shown for equation (4.20b).

Table 4.3: Comparison of trust-confidence measures. Total number of items is 1000, ratios are rounded to 3 decimals

$ I_x $	$ I_y $	$ I_{xy} $	$conf_{sim_1}$	$conf_{sim_2}$	$conf_{sim_3}$
10	10	10	0.01	1	1
10	100	10	0.01	0.182	0.1
100	100	10	0.01	0.1	0.053
100	100	50	0.05	0.5	0.333
100	100	100	0.1	1	1

Luckily, the sums used in this definitions of the confidence in user similarity are already calculated during the incremental update of the correlation itself, so the computational complexity here is also only in  $O(1)$ .

### 4.2.2 Trust model in Skipforward

The trust model we use in Skipforward is very similar to the model used for features, so the competence trust in another user can be seen as a non-binary feature type, where the applicability is equivalent to the user similarity and the confidence has the same meaning<sup>4</sup>. But in contrast to the applicability value, which has a range of  $[-1; 1]$ , we limit the range of the competence trust to non-negative values. The trust of user  $u_x$  in user  $u_y$  regarding features of type  $t_f$  can be described as a function

$$trust : U \times U \times T_f \longrightarrow [0; 1] \times [0; 1]$$

that maps every pair of users and a feature type to a trust value consisting of the raw competence trust and the confidence.

$$trust(u_x, u_y, t_f) \longmapsto (comp(u_x, u_y, t_f), conf(u_x, u_y, t_f)) \quad (4.21)$$

There are different possibilities to map the user similarity  $sim$  with range  $[-1; 1]$  to the desired range of  $[0; 1]$ , which is why we introduce a new function  $comp$  that can be defined in different ways to allow more control over the mapping, for example:

$$comp(u_x, u_y, t_f) = \max(0, sim(u_x, u_y, t_f)) \quad (4.22a)$$

$$comp(u_x, u_y, t_f) = \frac{(sim(u_x, u_y, t_f) + 1)}{2} \quad (4.22b)$$

In definition of the competence trust value in (4.22a) user similarities below 0 are simply interpreted as 0, whereas in (4.22b) the interval  $[-1; 1]$  is uniformly mapped to the interval  $[0; 1]$ , so that a user similarity of 0 still leads to a competence trust value of 0.5.

### 4.2.3 Feature aggregation

We define the feature aggregation function for a specific feature type  $t_f$  of a user  $u_x$  as:

---

<sup>4</sup>A special domain ontology, called *Skippies* ontology, that allows expressing opinions about users, such as trust and other features, is under development and will be added in a future version of Skipforward.

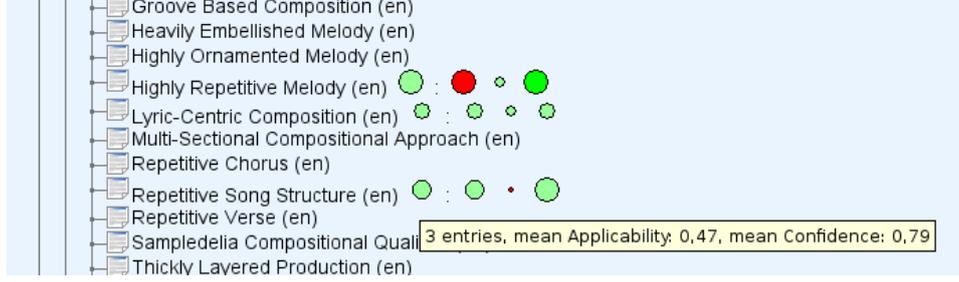


Figure 4.1: Screenshot of feature aggregation in Skipforward

$$aggr_{app}(u_x, i) = \frac{\sum_{u_y \in \hat{U}} comp(u_x, u_y) conf(u_x, u_y) c_{y,i} a_{y,i}}{\sum_{u_y \in \hat{U}} comp(u_x, u_y) conf(u_x, u_y) c_{y,i}} \quad (4.23)$$

$$aggr_{conf}(u_x, i) = \frac{\sum_{u_y \in \hat{U}} comp(u_x, u_y) conf(u_x, u_y) c_{y,i}}{\sum_{u_y \in \hat{U}} comp(u_x, u_y) conf(u_x, u_y)} \quad (4.24)$$

with

$$\hat{U} = \{u \in U \mid f(u, i) \neq \emptyset\} \quad \text{and} \\ f(u_y, i) = (a_{y,i}, c_{y,i})$$

where  $aggr_{app}$  is the aggregated applicability and  $aggr_{conf}$  the aggregated confidence. As these equations show, we define the aggregation as a weighted mean that includes the trust in another user into the weight, so the aggregated confidence is merely a mean of the confidence and not a sum, because it is hard to define the semantics of trust values (see Section 3.1.3): Should the aggregated confidence in a feature stated by two users with a confidence of 0.5 have the same weight as a feature made by one user with a confidence of 1?

Instead of imposing a calculation on the user that might be difficult to interpret we prefer to display these weighted means as aggregation along with a list of the actual features that led to this value (see screenshot in Figure 4.1).

#### 4.2.4 Using domain knowledge to reduce data sparsity

Even though Skipforward itself is not a Recommendation System, its database is a good foundation for applying content-based filtering algorithms. One of the main problems of recommendation systems is the sparsity of data, which also applies for Skipforward, but in different contexts.

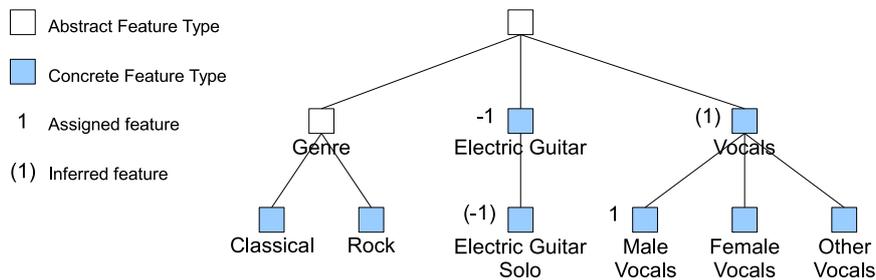


Figure 4.2: Inferring on Feature Types

First of all, sparsity in CF systems describes the problem that users have only rated a small fraction of the items in the database, thus leading to a small number of items commonly rated with other users. Skipforward uses CF methods to calculate the similarity between the opinions of two users regarding a specific feature type, so the sparsity problem applies for each of them separately.

The domain ontologies often define hundreds of different feature types<sup>5</sup> and it is unrealistic to assume that users express opinions about all of them.

The idea here is to utilize the domain knowledge contained in the ontologies to infer opinions that have not been explicitly expressed without making large requirements. The probably most intuitive inference that can be made uses the class hierarchy of feature types: If, for example, Alice states that a song has “male vocals”, one would intuitively reason that she also states that this song has “vocals”, because “male vocals” is a special case, i.e., a subclass, of “vocals”. In the same way, if Alice says that a song has no “electric guitar”, she implicitly claims that the song has no “electric guitar solo”.

This inference is only possible in the described directions, because if a song is said to not have “male vocals” it generally can not be reasoned that it does not have the feature “vocals”. Feature types that merely exist for grouping and organizing are abstract classes, as it would not make much sense to create features for them (see Figure 4.2), so they are excluded from the inference, too.

This inference can be generalized in the following way: If in the class hierarchy of a domain ontology every subclass of a feature type is a specialization of this feature type, then features with negative applicability can be inferred to subclasses, whereas features with positive applicability can be inferred to superclasses.

When there are multiple features of subclasses (superclasses) of a feature type that has no feature explicitly assigned, the inferred feature has the applicability and confidence of the one with the highest confidence and (if there are multiple features with highest confidence) the highest (lowest) applicability (see Figure 4.3 for an example).

<sup>5</sup>For example, Skiptrax, the music feature ontology used in the evaluation, contains more than 600 feature types and yet some users stated that some aspects of songs could not be expressed due to missing feature types.

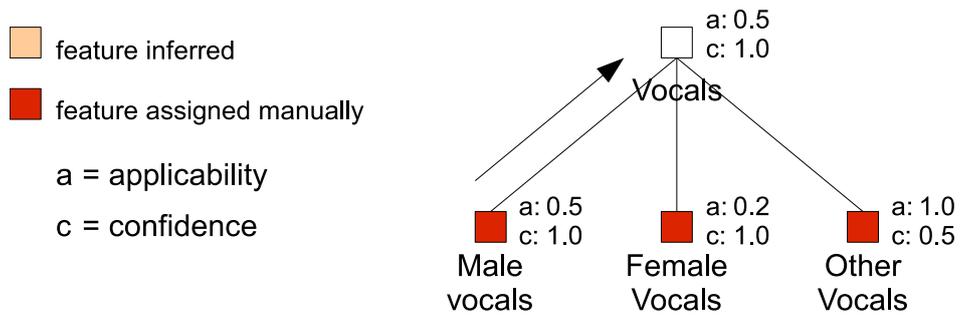


Figure 4.3: Inferencing with multiple features. The feature with highest applicability found in subclasses of “Vocals” does not have the highest confidence, so the feature with highest applicability of those with highest confidence is inferred upwards.

## Chapter 5

# Implementation

When integrating the algorithms developed in this thesis into Skipforward, we added it as an additional layer of functionality so that it is still possible to run Skipforward without the new components. The data for the similarity calculation are stored separately from the items and their features, allowing a better abstraction from the data format and making it easier to develop and test the developed algorithms. The data repository of the SimilarityManager will be called *cache* from here on.

The traditional way of calculating the user similarities from scratch can easily be accomplished by querying the Skipforward API for all items and their features, but in order to be able to incrementally update the similarities changes to the database need to trigger the calculation somehow.

Changes to the database of a Skipforward node are made in two different ways: Firstly, by using the provided API through some user interface or other program, and secondly, through direct manipulation of the underlying RDF model by the Syncer component. Thus, it is not sufficient to modify the data changing methods of the interface, but instead we have to listen to the RDF model directly to get notified of changes.

The SimilarityManager, as we call the component that manages user similarities, listens for changes on the model and calculates the necessary updates when it detects a change. Skipforward itself then can query those similarities and use them to personalize feature aggregations or compute recommendations.

### 5.1 Main algorithms

#### 5.1.1 Initializing the similarity values

If a new user logs in into Skipforward for the first time, his database contains only the Skipinions ontology and maybe some other domain ontologies. Starting without any features, all values that have to be cached for the incremental update of the user similarities are 0 or undefined, so these values can be initialized lazily

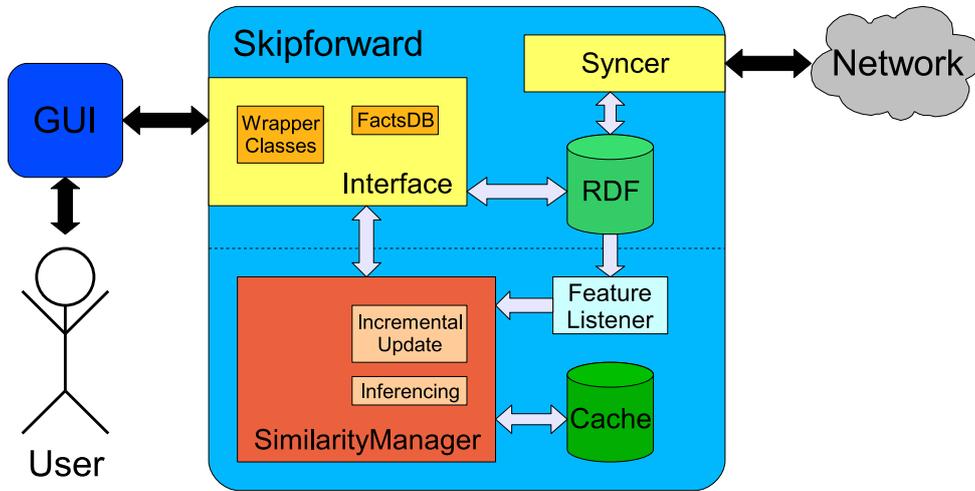


Figure 5.1: Architecture of Skipforward with SimilarityManager

when they are needed during an incremental update.

However, there can be many situations in which the cached values and similarities need to be recalculated, for example when parameters for the correlation have changed, like the definition of the combined weight.

**Algorithm 5.1.1:** CALCULATEALLCORRELATIONS()

```

for each  $i \in I$ 
   $copyFeaturesToCache(i)$ 
  for each  $u \in U$ 
     $doInferencing(u, i)$ 
  for each  $t_f \in T_f$ 
     $calculateMean(localuser, t_f)$ 
  for each  $u \in U$ 
     $calculateMean(u, t_f)$ 
     $calculateCorrelation(localuser, u, t_f)$ 

```

Before the user similarities are calculated, the *valid features* have to be identified. Since it is possible for a user to create a feature for an item that already has a feature assigned through him, only the newest feature of each type for each item is valid. These valid features are then copied to the cache and the inferencing rules described in Section 4.2.4 are applied for each item and user. The inferred values are also stored in the cache, increasing the number of available feature data.

The calculation of the mean applicability is a simple iteration over all items with features of type  $t_f$  from the corresponding user described in equation (4.8). For each friend of the local user, the correlation between the local user and him is

calculated according to equation (4.7), which essentially is an iteration of all items that have features of type  $t_f$  assigned by both users (explicitly or by inference).

During the calculation of the mean and the correlation, all values required by the incremental update of user similarities can easily be obtained without a change in the computational complexity, since the number of iterations does not change.

### 5.1.2 Listening for feature changes

The FeatureListener is an implementation of the Jena *ModelChangedListener* interface that registers to the RDF model of Skipforward's database and then gets notified of every RDF statements that are added or removed from the model. By the time of notification, the changes to the model have already been made. For the update of user similarities, however, it is necessary to not only know the new valid feature values, but also the feature values that were valid before that.

For example, if a new feature was added, the notification of the FeatureListener contains the information about this new feature, but none about if it made a previous feature invalid and if it did, what applicability and confidence values the previous one had. This information can be gathered by getting all features for this item (with the same feature type and from the same user) and looking for the one with the latest creation date and that is not the one added.<sup>1</sup>

Although the necessary queries on the RDF model are not hard to implement, the performance heavily depends on the way the used RDF framework processes these queries. Since every single feature change requires the lookup of the previously valid feature values, this approach could easily increase the computation time for similarity updates.

A faster solution to retrieve the old feature values is to get them from the cache of the SimilarityManager. When initialized, the cache was filled with all valid feature values, the ones assigned by the user and the inferred ones. Hence, when the RDF model has been changed, the required information about its previous state can be queried from the cache. However, after all user similarities have been updated according to the *feature changes*, the feature values in the cache have to be updated, too.

### 5.1.3 Incremental update of similarity values

Every single feature change is handled separately and requires an update of one or more user similarities, depending on who the changed feature belongs to. Since we only maintain the user similarities between the local user and his friends, only one update is necessary when the feature of another user has changed, but the similarities to all other users have to be updated if a feature of the local user has changed (see Figure 5.2).

---

<sup>1</sup>As a real world example, imagine your bank sends you a notification and tells you that the new balance of your account is 500 EUR. If you are interested in the difference to the previous balance (like we are here) you have to search for the latest balance notification before this new one.

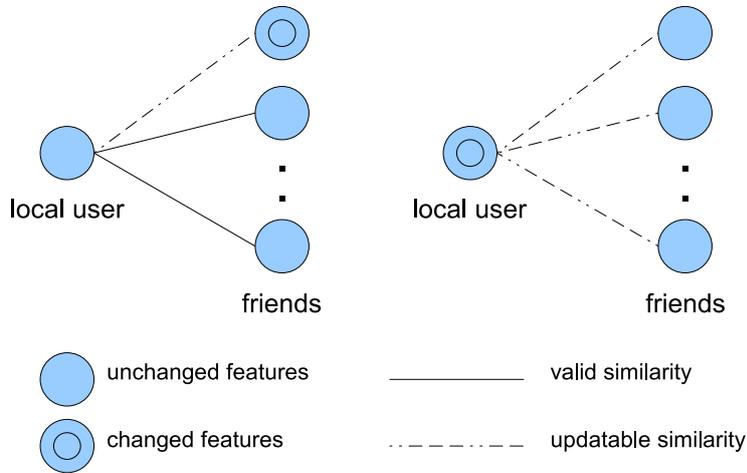


Figure 5.2: Updates after a feature change

**Algorithm 5.1.2:**  $UPDATESIMILARITIES(u_a, t_f)$

```

updateMean( $u_a, t_f$ )
if ( $u_a = localuser$ )
  for each  $u \in U$ 
    updateCorrelation( $u_a, u, t_f$ )
else
  updateCorrelation( $u_a, localuser, t_f$ )

```

The methods for incrementally updating mean and user similarity calculate the changes based on the formulas described in Section 4.1.4. The data required for this calculations are most of the cached values as well as the old and the new values for the mean and the feature that has changed.

But although the feature changes are handled separately, they do not always occur separately. For example, when the Syncer receives an update of the model of another user, all changes to the local RDF model are applied at once, thus adding or removing multiple features. It can also happen that the model update contains multiple feature from one user for the same item and same feature type, because he added a new feature and then revised his opinion with a newer feature.

Additionally, every change of features means that inferred features may no longer be valid, so for every item for which a feature change occurred the inferencing has to be repeated. If the inferencing is done directly after each feature change and these changes are processed in an arbitrary order, an item might be processed several times, which is unnecessary (see Figure 5.3).

A more efficient way is to group feature changes by item and user and copy the new feature values to the cache. Then, inferencing for this item and user is done only once, updating more feature values in the cache (see Figure 5.4). It should

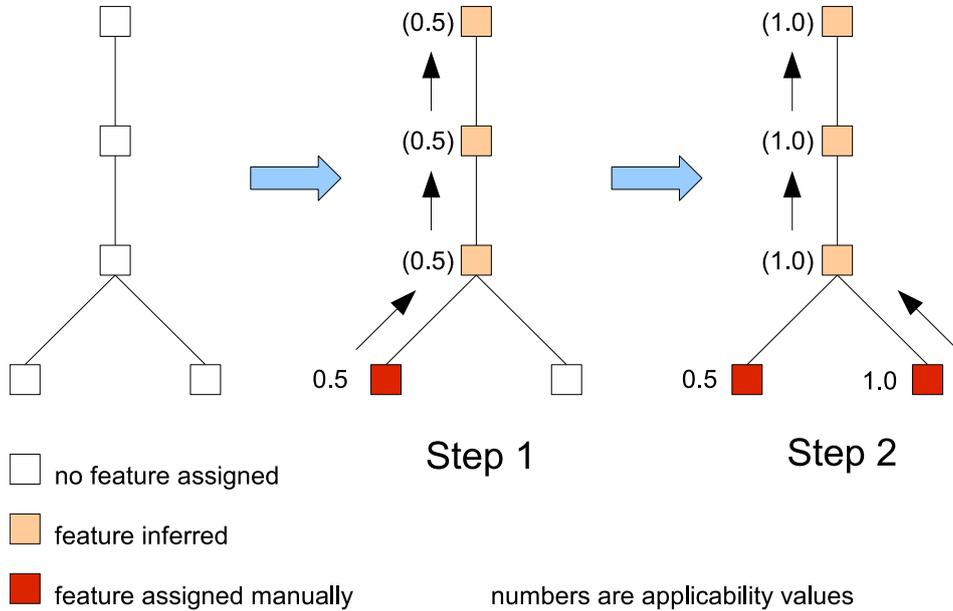


Figure 5.3: If the inferencing is started after each update, it is possible that feature values are inferred that will be overwritten again, which is unnecessary.

be noted, that before a cached feature value is updated in the cache, it is stored in a list, preserving the previous state that else would be lost. After all values have been inferred, user similarities for the types of changed features, through explicit update or inferencing, are incrementally updated by getting the old feature values from the list and the new values from the cache.

**Algorithm 5.1.3:** UPDATESIMILARITIES()

```

groupFeatureChanges()
for each  $u \in \text{changedGroup}$ 
  for each  $i \in \text{changedItems}(u)$ 
     $\text{changedList} = \{\}$ 
    for each  $t_f \in \text{changedFeatures}(u, i)$ 
       $\text{changedList.add}(\text{getFeature}(u, i, t_f))$ 
       $\text{copyChangesToCache}(u, i, t_f)$ 
    }
     $\text{doInferencing}(u, i)$ 
     $\text{updateMeanAndCorrelation}(u, i)$ 

```

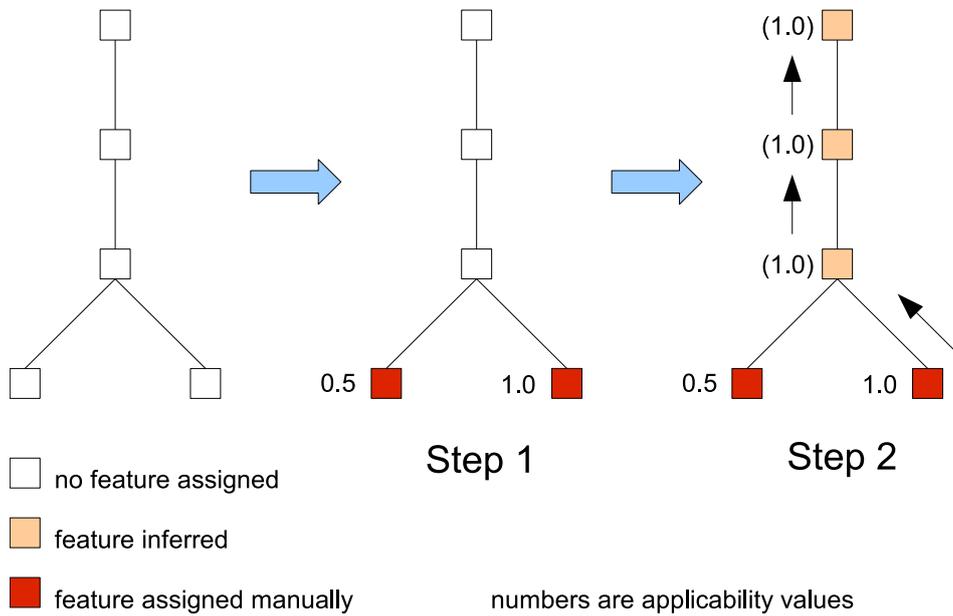


Figure 5.4: If inferencing is started after all explicit changes have been committed, unnecessary feature value assignments can be avoided.

### 5.1.4 Inferencing

When a feature change occurs, it is not necessary to check the inferred values for all feature types, but only for those that may be affected, i.e., the direct sub- and superclasses of the feature type where the change occurred. If one of those changed by the inferencing, its direct sub- and superclasses need to be checked, too, and so on.

**Algorithm 5.1.4:** UPDATESIMILARITIES()

```

updateSet = {}
for each  $t_f \in \text{featureChanges}$ 
  updateSet.addAll( $t_f.\text{directSubClasses}()$ )
  updateSet.addAll( $t_f.\text{directSuperClasses}()$ )
while not updateSet.isEmpty()
  {
     $t_f = \text{updateSet.getFirstElement}()$ 
    updateSet.remove( $t_f$ )
    if  $t_f.\text{isAbstract}()$  or  $t_f.\text{isExplicitlySet}()$ 
      then continue
  }
do {
  else {
     $f = \text{inferFeatureValue}(t_f)$ 
    if  $f \neq \text{currentFeature}(t_f)$ 
      then {
        updateSet.addAll( $t_f.\text{directSubClasses}()$ )
        updateSet.addAll( $t_f.\text{directSuperClasses}()$ )
        setCurrentFeature( $t_f, f$ )
      }
  }
}

```

Essentially, everytime a feature is changed, the direct sub- and superclasses of its feature type are added to a list of feature types that may need to be updated. Then, a feature type of this list is removed and if it is not abstract or explicitly set, a new feature value is inferred with the rules described in Section 4.2.4. If this feature value is different from the current one (if one exists), the new one is set and its sub- and superclasses are again added to the list. This is repeated until the list is empty.

## 5.2 Data structures

To be able to implement the approaches described in Chapter 4, different types of data need to be stored. The cache mainly contains two kinds of data: the valid features and the values necessary for incremental calculation of user similarities.

### 5.2.1 Feature Cache

As already mentioned in Section 5.1.2, caching the feature values can speed things up whenever the currently valid feature value needs to be queried, so the data structure should be fast. On the other hand, the knowledge we have about the nature of the data should be used to reduce memory consumption.

#### What needs to be stored?

Since the calculation of user similarities is based on calculation done in collaborative filter, it is suggesting to base the data structure for the feature values on the user-item matrices (see Section 3.2.1). The user-item matrix in CF stores the ratings users assign to items and the approach of this thesis is to view every single

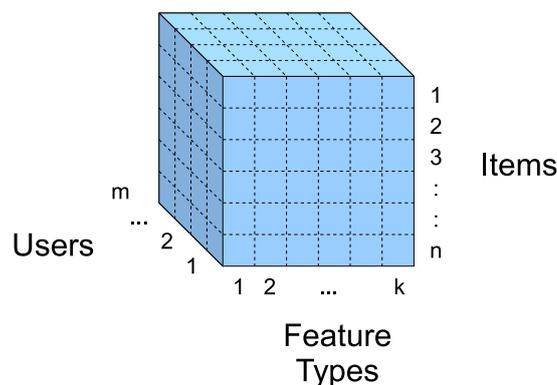


Figure 5.5: User-Item-FeatureType Matrix

feature type as its own type of rating. Consequently, the feature values in Skip-forward can be represented by a 3-dimensional user-item-featuretype matrix (see Figure 5.5).

There are many possibilities to represent a 3-dimensional matrix and each has its pros and cons, so we need to examine the properties of the data that are to be stored.

**Data sparsity** The matrix will be mostly empty due to the sparsity of data. A simple 3-dimensional array would not be suitable here, because the memory needed to store the array is allocated at its creation time, even if most of the cells in the matrix stay empty.

**Access order** Feature values may be accessed in different succession, such as item-by-item during inferencing, or one feature type at a time when initializing user similarities, so the data structure has to be fast regardless of the order in which elements are accessed.

**Variable size** The three dimensions have variable size, since users, items and feature types can be added or removed from the database at any time

**Element addressing** Feature values are addressed by an URI for each dimension, not integer numbers. Additionally, the URIs for items differ from user to user, because they have to be in his namespace.

### Sparse matrices and element addressing

In our implementation we use the Colt<sup>2</sup> library, because it was developed to work with large datasets with respect to performance and efficiency and provides a stable release. In particular, it offers implementations of sparse matrices, where cells that are zero do not consume memory and access to cells has an expected complexity

<sup>2</sup><http://acs.lbl.gov/~hoschek/colt/>

of  $O(1)$ . Therefore, it fulfills our first two criteria, low memory consumption for sparse matrices and fast access time in any order.

The size of these matrices is fixed and has to be specified at creation time, so we need to simply create the matrix as large as possible, since empty cells do not use memory.

The biggest problem that has to be solved is the way in which cells are addressed, because most matrix implementations only allow integer values for indices. The values that are used in Skipforward to uniquely address a valid feature are all complex objects, but they can be reduced to unique string representations: Each item or feature type can be uniquely identified by its URI, and each user can be uniquely identified by his account name or namespace, which all are strings.

Strings can not be used as indices for the matrix, so we have to define a mapping from strings to integer indices. This mapping can not be realized through a hash function, because the number of valid integers is very small compared to the number of possible URIs and it is not known in advance which URIs will be used.

Therefore, the mapping has to be established on-demand: If the integer value representing the index of a specific URI is requested but not defined, an unused integer value will be assigned to it and stored. The next time the index for this URI is queried, the previously assigned integer value will be returned.

Since the matrix and therefore its dimensions have fixed size, the available integer indices are limited and thus the number of users, items and feature types that can be addressed, too. If all available indices for items are already used, new items can not be addressed and feature values for them can not be stored, even if there is plenty of space in the matrix.

The next problem related to URIs is the identity of items: As described in Section 2.4, every user describes only items in his own namespace and the transitive *isIdenticalTo* relation is used to define that two resources of different namespaces actually refer to the same item. Consequently, each item may have multiple URIs referring to it. So, either the mapping from URIs to integer values has to support multiple URIs mapping to the same value, or it has to be ensured that each item is always identified by the same URI. We chose the second approach, because mapping multiple URIs to one integer value would require us to keep track of changes in *isIdenticalTo* relations, which makes things more complicated and error prone.

Instead of having one 3-dimensional user-item-featuretype matrix for everything, we create a separate 2-dimensional item-featuretype matrix and a separate item-index mapping for each user. Since there are only feature values from one user stored in this matrix, all items this user has assigned features to have an URI in his namespace. Consequently, if there is no URI for an item in the user's namespace, he has not assigned any features to it and there are no values to be stored in this matrix.

Additionally, the memory is used more efficiently, because each matrix contains only rows for the items its user has assigned features to and the item-index mapping maps only URIs of the user's namespace.

However, when querying items that have been co-rated, i.e., two users have assigned features of the same type to it, this is not evident from the matrix and the mapping alone. For each user, the items with features of this type in his namespace have to be retrieved and the semantically identical ones have to be found by querying the RDF store.

### 5.2.2 Incremental user similarities

The incremental calculation of user similarities requires the caching of values for each feature type and for each user and each pair of users for which a similarity should be calculated. Examples for user-specific values are the mean applicability and the sum of confidence values of features of type  $t_f$ , whereas the factors  $B$ ,  $C$  and  $D$  have to be stored for each pair of users (see Section 4.1.4 for a complete list).

These values can be stored in hash maps or other simple data objects. To increase memory efficiency, these data objects are instantiated lazily, because the values for many feature types are often 0 if there are no co-rated items with other users. Instead of storing these “empty” values in an object, no object is created, which is interpreted by the system as “all values are 0”.

## Chapter 6

# Evaluation

The main focus of this diploma thesis is to develop performant trust and similarity metrics, but the efficiency of the algorithms and of their implementation would be useless if they did not produce reasonable results. Therefore, we tested them with data gathered in an evaluation with five participants that were given the task to annotate a certain number of songs with a given set of feature types from the Skiptrax ontology. Section 6.1 describes the parameters of this evaluation and statistical results for gathered data are shown in Section 6.2.

Additionally, automated tests were made to compare the time needed by our algorithm for an incremental update and the classical non-incremental calculation of user similarities, and are presented in Section 6.3.

### 6.1 Evaluation parameters

The domain ontology used for this evaluation is the Skiptrax ontology. It is based on a list of features used by Pandora (see Page 4), which can be found at the English Wikipedia website<sup>1</sup>.

50 different songs were selected and presented to five participants. Each participant had to select 25 songs randomly, which they had to describe with a set of 42 binary feature types<sup>2</sup> that were chosen from the 650 feature types of the Skiptrax ontology. With these parameters followed it was guaranteed that there would be multiple songs that were annotated by at least two or three persons, providing enough data to be able to calculate user similarities.

### 6.2 Statistical results

There are many different ways to calculate user similarities that were presented in Chapter 4 and we can choose between

---

<sup>1</sup>[http://en.wikipedia.org/wiki/List\\_of\\_Music\\_Genome\\_Project\\_attributes\\_by\\_type](http://en.wikipedia.org/wiki/List_of_Music_Genome_Project_attributes_by_type)

<sup>2</sup>A list of these feature types can be found in Appendix C

- *normal* and *constrained* Pearson correlation,
- *minimum* and *geometrical mean* for the combined weight, and
- *user-specific* or *combined* weights in the denominator

leaving us with eight different combinations. We will only present statistics for four of these possible combinations that we consider significant and interesting. With 42 feature types and 5 users, we have  $(5 * 4 * 42)/2 = 420$  trust values that we could examine, but which would not make much sense. Instead, we will plot all trust values into one 3-dimensional and two 2-dimensional plots, where the height of the graph represents the number of trust values in the corresponding range.

Figures 6.1 and 6.2 show the distribution of trust values of the normal Pearson correlation, and Figures 6.3 and 6.4 show the constrained Pearson correlation, whereas in both cases the first plots are the correlations using the combined weights, and the second plots the user-specific weights. All four correlations use the minimum as combined weight (equation (4.3)) and the correlation value is mapped uniformly to the interval  $[0; 1]$  (see equation (4.22b)). The confidence in the similarity is always calculated using equation (4.20b) and is therefore the same in each plot.

One can see that the plots with normal and constrained correlation are very different from each other. Besides the fact that the normal Pearson correlation leads to 83 undefined values (which are not included in the plots here), it generally results in less values near a similarity of 1.0 and more values near 0.5. The difference between combined and user-specific weights is marginal in both cases, although greater in the case of the normal Pearson correlation.

Since a high similarity to another user results in a greater influence of his features on the personalized view, we decided to use the constrained Pearson correlation for calculating trust values. Figure 6.5 shows a section of the similarity view from two users of the evaluation. One can see that users have always full similarity and confidence to themselves.

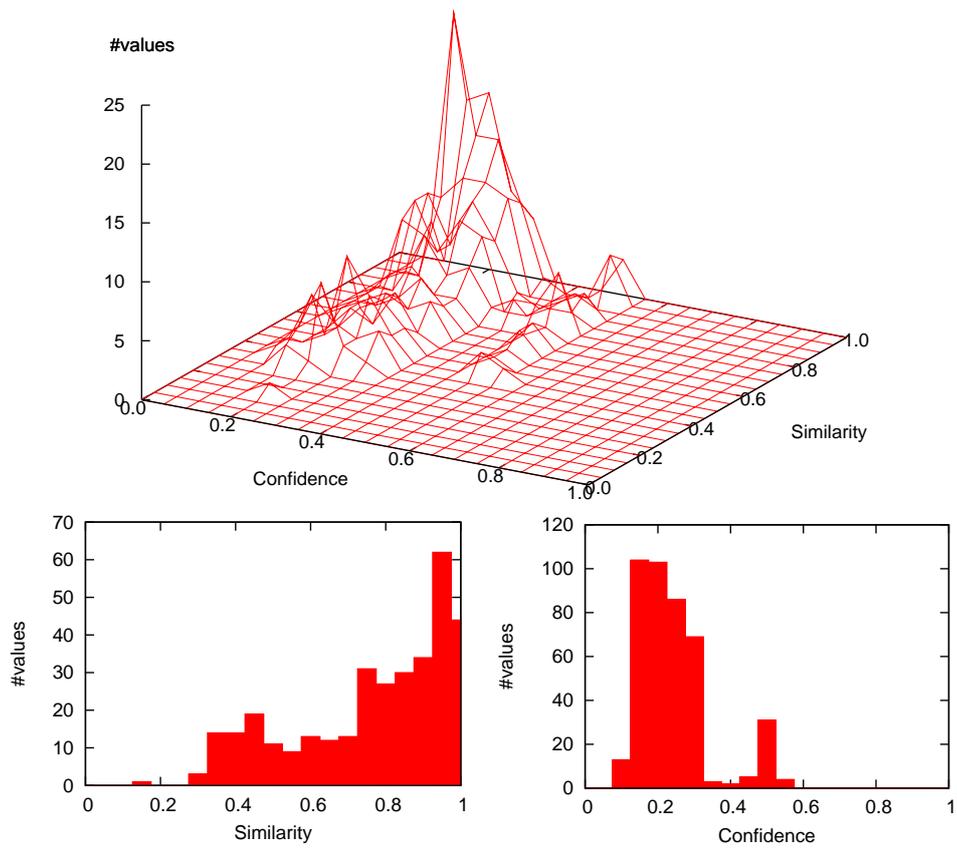


Figure 6.1: Statistics of trust values with **combined** weights (eq. (4.6)) and **normal** Pearson correlation. 83 correlations are undefined and would have been mapped to 0.5, but are left out here for reasons of better visibility for the rest of the data.

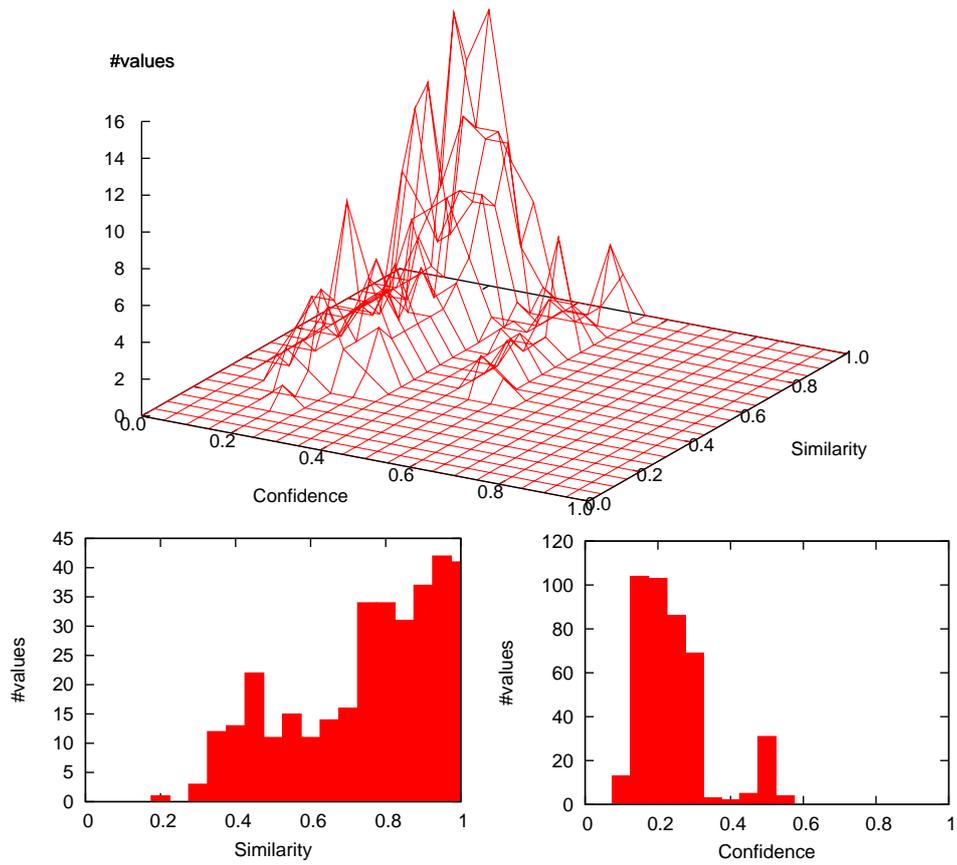


Figure 6.2: Statistics of trust values with **user-specific** weights (eq. (4.5)) and **normal** Pearson correlation. 83 correlations are undefined and would have been mapped to 0.5, but are left out here for reasons of better visibility for the rest of the data.

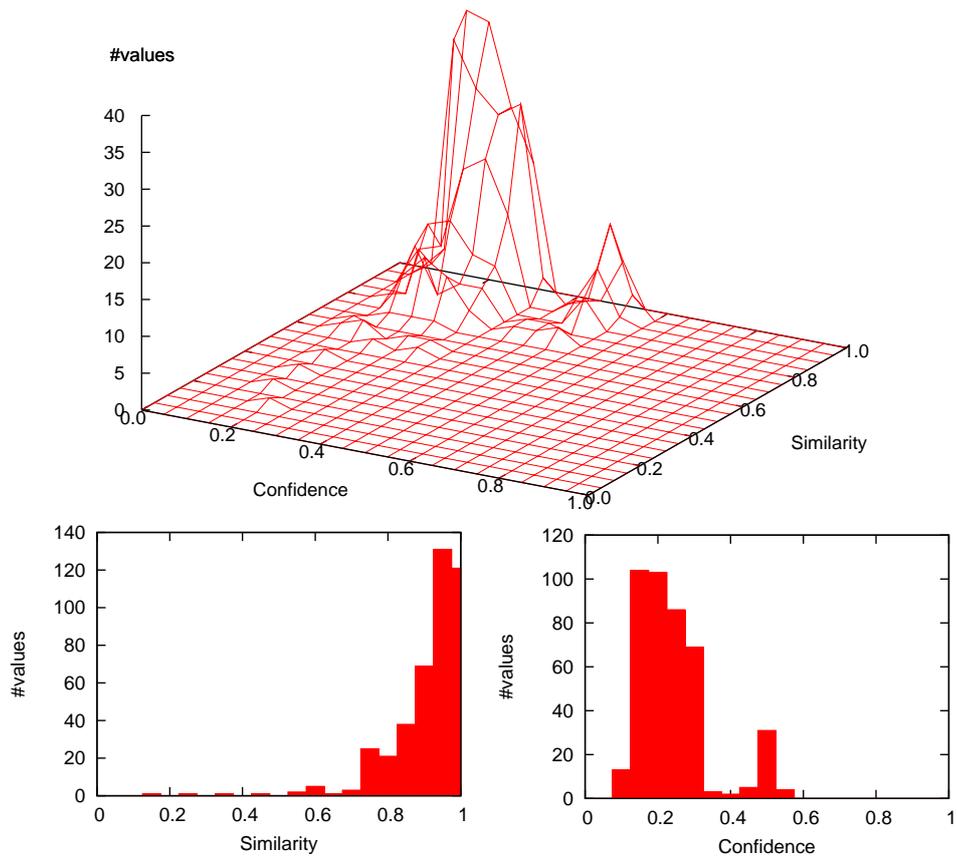


Figure 6.3: Statistics of trust values with **combined** weights (eq. (4.6)) and **constrained** Pearson correlation. No correlations are undefined.

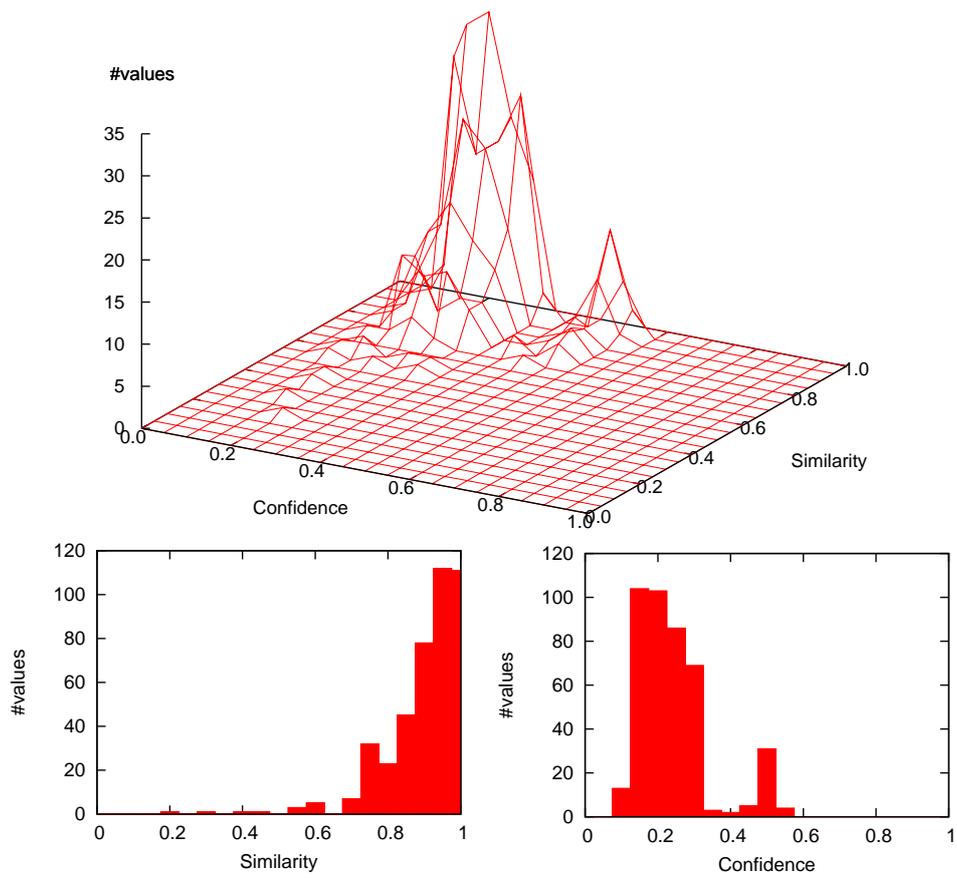
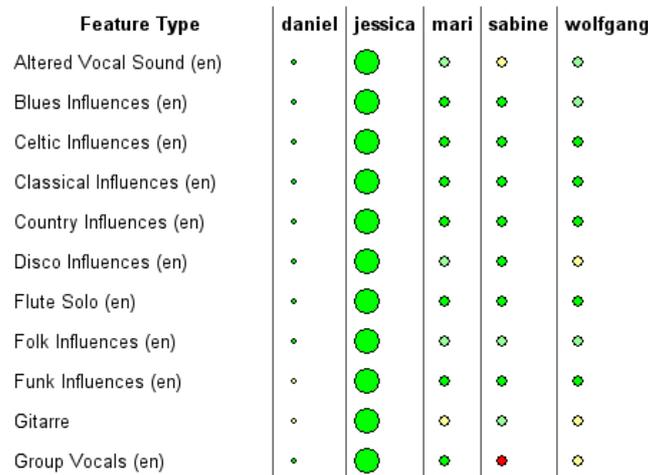


Figure 6.4: Statistics of trust values with **user-specific** weights (eq. (4.5)) and **constrained** Pearson correlation. No correlations are undefined.

## Similarities (Correlation)



## Similarities (Correlation)

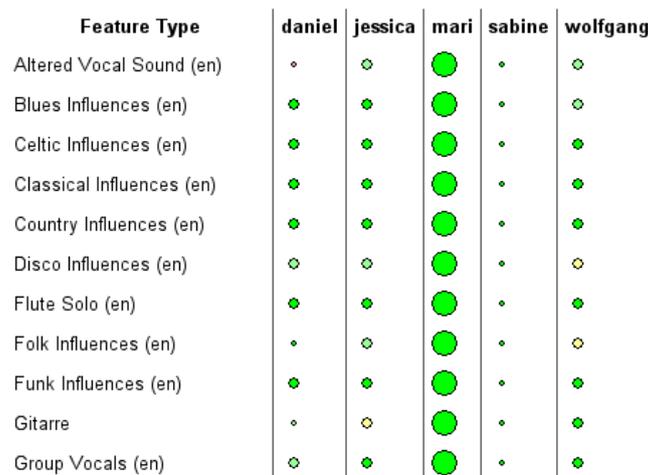


Figure 6.5: Similarity overview from two participants of the evaluation. Like the circles for feature values, this circles represent the trust the user has into another user concerning the different feature type. Bigger circles mean higher confidence and the similarity value ranges from red for  $-1$  over yellow for  $0$  to green for  $1$ . It is obvious that the upper graphic belongs to user “jessica”, because there is full trust with full confidence for all feature types. Her trust in “daniel” is based on fewer data than her trust to the other users, which is why these circles are smaller representing a low confidence in the calculated similarity. User “mari” on the other hand, whose trust values are shown in the lower graphic, can base his similarity to “daniel” on more information, hence the confidence is higher. Note, that the trust between “jessica” and “mari” is the same, since similarity and confidence are defined symmetrically.

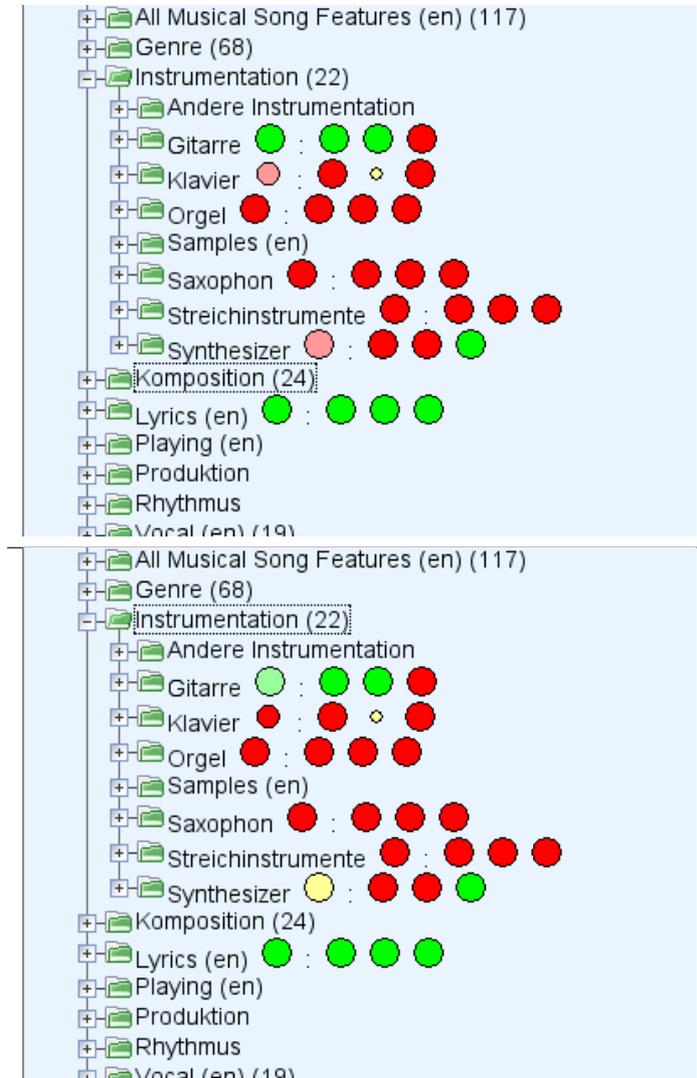


Figure 6.6: Different views on the same item. Both screenshots show the same item from the point of different users. Due to different trust in other users, the features are aggregated differently, as can be seen from the color of the circles: The aggregation of the three features for “Synthesizer” in the upper screenshot is between neutral and negative, as can be seen from the light red color. The same three features, two fully positive and one negative, are aggregated to a neutral value in the lower screenshot, indicating that this user trusts the one positive feature more than the two negative ones.

#items	#changes	1 friend		5 friends		10 friends		20 friends	
		all	incr.	all	incr.	all	incr.	all	incr.
10	1	76	1	79	3	215	5	222	4
	10	153	10	174	30	361	69	366	57
100	1	23	1	69	1	183	1	452	3
	10	119	3	397	7	970	12	2449	29
	100		32		85		125		269
1000	1	46	1	240	2	739	3	2593	6
	10	1831	6	11352	17	31322	27	100234	59
	100		68		159		281		590
	1000		940		1513		3011		6385

Table 6.1: Performance comparison of non-incremental and incremental similarity calculation. All values are milliseconds (ms).

### 6.3 Performance

The performance of the incremental and the non-incremental is compared for different numbers of items, users and feature changes. At first, the database of Skipforward is populated with items for the active user and for each friend of this user the corresponding “identical” items are created. Then, for each user a feature of a fixed type is created for each item. Inferencing is disabled and only one feature type is used.

When calculating user similarities from scratch, it has to be iterated over all items no matter how many features have changed. Also, if no values have been cached, the currently valid features have to be extracted from the RDF store before the actual calculation can begin. This is exactly what is being done during the initialization of the SimilarityManager, so this method is called and its execution time measured.

With the database being populated and all values necessary for the incremental calculation stored, feature changes are made in batches of different size for the active user and the time needed for the incremental update is measured.

This measurements are repeated four times and the minimum values are taken to reduce the influence of fluctuations caused by the scheduler of the operating system. The resulting values are shown in Table 6.1 in milliseconds (ms).

At first we examine the time needed by the incremental algorithm: If we look at each setting of items and friends individually, we notice that the execution time grows approximately linearly with the number of features that have changed (right columns), which is expected if we assume that each similarity update needs constant time. Also, we notice that the execution time stays nearly the same if the total number of items is increased, but grows with the number of friends. This observation too fits the expectation of constant execution time, since each feature change of the active user causes the user similarities to all friends being updated.

The time needed by the non-incremental algorithm is shown in the left columns, the values in normal letters are the times for calculation only, whereas the italic values are the times needed for complete initialization process. These values have a higher fluctuation than those of the incremental updates, but they generally tend to grow with the number of items and friends.

One may notice that the execution time of the initialization method does not increase linearly but stronger. A possible reason for this becomes apparent when looking closer at the process of populating the RDF database. As it turns out, the time needed to add items and features to the database increases with the size of the database. If the access time of the RDF store is not constant, this might slow down the calculation of the user similarities as well.

It should also be mentioned, that with an increasing number of total items, the incremental update becomes less performant for larger feature changes compared to the non-incremental calculation. This is not surprising, since the equations for the incremental update are rather complex and when a certain amount of feature changes is exceeded, it is more performant to do a non-incremental calculation of the similarity values.

## Chapter 7

# Summary and Outlook

In this thesis, we have developed incremental algorithms to calculate similarities between users based on opinions they have expressed. We were able to use these similarities as trust values to create a personalized view on a knowledge-base that is inherently inconsistent. These personalized views provide a useful foundation to build mechanisms upon it that help the user in his task of identifying relevant information in large data sets.

The algorithms are based on well established methods in the field of Recommender Systems and combine aspects of both collaborative and content-based filtering. The incremental calculation of user similarities during the usage of Skip-forward ensures that the data necessary for the aggregation of features and recommendation of interesting items is always updated and at hand.

We evaluated the results of our algorithms on the basis of annotations of music of five persons collected in a test period of three weeks and the gathered data was sufficient to indicate the plausibility of the results. Nevertheless, an extended survey with more participants should be performed to verify the quality of the personalization of data.

Experiments regarding the performance of the incremental algorithms have shown that they are indeed faster than the classical non-incremental calculation of similarities for many cases. Also, the absolute time needed for the incremental updates on an ordinary computer is in the region of seconds, making it practicable for use on personal computers.

### 7.1 Outlook

#### 7.1.1 Optimization of data structures

For the implementation of our algorithms we used the Jena and the Colt framework for storage, because it allowed us to concentrate on the main tasks of this thesis. However, during the implementation we encountered some complications because of the different ways in which data is addressed. The mapping from URIs to integer

values as indices and vice-versa does not only consume memory, but it also takes some time to do the mapping.

Instead of translating between both frameworks, it should be investigated if more efficient ways for storing the data required for our algorithms can be developed.

### **7.1.2 Relationships between feature types**

Currently, the only relationship between feature types that is used in the calculations is the subtype relationship. This allows us to infer some features that the user implicitly defined by explicitly assigning another feature. The feature type hierarchy is defined in the domain ontologies and the creator of the ontology is responsible for designing these relationships properly.

There may be dependencies between different feature types that were not apparent to the ontology creator or that are not generally valid for all users. Different techniques for discovering these relationships are possible. For example, the already used correlation coefficient could be used to examine dependencies between feature types. This or other approaches might be suitable to further increase the information that can be automatically extracted from the user's inputs.

### **7.1.3 Handling of non-binary feature types**

The user similarities used in this thesis were based on binary feature types only, reducing the complexity of possible similarity functions. However, valuable information is ignored when non-binary features are not included in this similarities.

To some extent, non-binary feature types are replacements for a whole set of binary feature types. For example, instead of having dozens of different binary feature types for different kinds of musical influences, e.g., "Disco influences", "Jazz influences", "Rap influences", there could be only one non-binary feature type "hasInfluence" whose feature instances point to objects or items representing different influences.

In fact, this is sometimes the only possibility to represent certain features, because the number of available objects is infinite or unknown. For example, instead of using the feature type "MainArtist" that points to an item of type Artist, it would be possible to create an extra feature type for every artist available, like "MainArtist is Madonna", "MainArtist is Tina Turner", etc. However, this would massively increase the number of feature types for the same amount of information and therefore the sparsity of the data would also be increased.

There are some aspects regarding non-binary features that have to be further examined:

### **Similarity of objects**

If two users have assigned a feature of the same type to the same item, but the feature have different object, e.g. “MainArtist” with different artists, what is the similarity between these features? It could be as trivial as defining the similarity to be 1 if the objects are identical and 0 else. But depending on the type of the object, there could be a special similarity metric defined for it, e.g., the Hamming-distance for strings.

In either case, the algorithms developed in this thesis can not be applied to non-binary features without modification and the data structures used are not suitable in their current form, either.

### **Similarities on non-binary feature types**

The easiest way to define user similarities on non-binary feature types would be to have a separate similarity for each combination of feature type and object. For example, if two users agree that “Madonna” is an artist of a certain song, but disagree whether “Britney Spears” is an artist of this song, too, they would have a high similarity on opinions regarding Madonna being the artist of a song and a low similarity for Britney Spears. But it is not intuitively clear what is their similarity on the feature type “Artist” in general.

Further work is needed to define plausible rules and algorithms that process this information in a useful way.

#### **7.1.4 Similarities on partial data sets**

As already mentioned in Section 1.1, users can have different opinions on a certain topic while agreeing on another. Because user similarities are calculated separately for each feature type, a low similarity on one feature type does not influence the similarity of opinions of another type, allowing a very differentiated view of the competence of another user. This, however, is only because most feature type are restricted to a specific domain, e.g., “Strong melody”.

When a feature type can be applied to a wide range of item types, a differentiation of user similarities can not easily be accomplished. A good example for this problem is the feature type `skipinions:Review`, which is used to express ones liking of an item and can be applied to items of different types. However, for each pair of users only one value for the similarity on this feature type is calculated. This means that if two users have the same taste regarding music but enjoy totally different movies, the calculation will not distinguish between those item types and result in an intermediate similarity value.

# Appendix A

## Glossary

$U$ :	set of all users
$I$ :	set of all items
$I_x = \{i \in I   r_{x,i} \neq \emptyset\}$ :	set of all items rated by user $u_x$
$I_y = \{i \in I   r_{y,i} \neq \emptyset\}$ :	set of all items rated by user $u_y$
$I_{xy} = I_x \cap I_y$ :	set of all items rated by user $u_x$ and $u_y$
$T_f$ :	set of all features types

**Correlation** In general, the degree of relationship between two variables or functions. When viewing the opinions of a user  $u_x$  regarding a specific feature type  $t_f$  about all items as samples of a random variable  $X$ ,

$$\text{corr}(u_x, u_y, t_f) : U \times U \times T_f \longrightarrow [-1; +1] \subset \mathbb{R}$$

is the degree of correlation between the opinions of user  $u_x$  and user  $u_y$  regarding the feature type  $t_f$ , where a correlation of  $-1$  means that they always have opposite opinions,  $+1$  means that they have always the same opinion, and  $0$  indicates no relationship between them.

**User similarity** The similarity of the opinions of user  $u_x$  and user  $u_y$  regarding the feature type  $t_f$ , where a similarity of  $1$  means complete similarity and  $-1$  means no similarity.

$$\text{sim}(u_x, u_y, t_f) : U \times U \times T_f \longrightarrow [-1; +1]$$

**Competence** In general, competence is the ability to fulfill a specific task or function successfully. In the context of Skipforward:

$$\text{comp}(u_x, u_y, t_f) : U \times U \times T_f \longrightarrow [0; +1] \subset \mathbb{R}$$

The competence of user  $u_y$  to annotate items correctly regarding features of direct type  $t_f$  relative to user  $u_x$ .

**Expert** A user with a high competence in some domain. Due to the relativeness of competence, the role of an expert is also defined relative to another user. User  $u_x$  calls user  $u_y$  an expert, if

$$comp(u_x, u_y, t_f) \geq thresh_{u_x}$$

where  $thresh_{u_x} \in [0; +1]$  is a threshold defined by user  $u_x$ .

**Neighbor** The term neighbor is used in the context of collaborative filtering and describes users that have similar taste. It is a special case of an expert, where the feature type  $t_f$  is some kind of rating and, therefore, similarity and competence relate to the users taste.

## Appendix B

# Proof of equations

Note:  $I'_{xy}$  is the set of co-rated items without the active item (the one for which the feature change appeared).

$$\begin{aligned} B' &= \sum_{i \in I_{xy}} w'_{xy,i} (x_i - \bar{x}') (y_i - \bar{y}) \\ &= w'_{xy,i} (x'_a - \bar{x}') (y_a - \bar{y}) + \sum_{i \in I'_{xy}} w_{xy,i} (x_i - \bar{x}') (y_i - \bar{y}) \\ &= (w_{xy,a} + dw_{xy,a}) (x_a + dx_a - \bar{x}') (y_a - \bar{y}) + \sum_{i \in I'_{xy}} w_{xy,i} (x_i - \bar{x}') (y_i - \bar{y}) \\ &= (w_{xy,a} + dw_{xy,a}) (dx_a (y_a - \bar{y}) + (x_a - \bar{x}') (y_a - \bar{y})) \\ &+ \sum_{i \in I'_{xy}} w_{xy,i} (x_i - \bar{x}') (y_i - \bar{y}) \\ &= dw_{xy,a} (x_a + dx_a - \bar{x}') (y_a - \bar{y}) + w_{xy,a} dx_a (y_a - \bar{y}) + w_{xy,a} (x_a - \bar{x}') (y_a - \bar{y}) \\ &+ \sum_{i \in I'_{xy}} w_{xy,i} (x_i - \bar{x}') (y_i - \bar{y}) \\ &= dw_{xy,a} (x_a + dx_a - \bar{x}') (y_a - \bar{y}) + w_{xy,a} dx_a (y_a - \bar{y}) + \sum_{i \in I_{xy}} w_{xy,i} (x_i - \bar{x}') (y_i - \bar{y}) \\ &= dw_{xy,a} (x_a + dx_a - \bar{x}') (y_a - \bar{y}) + w_{xy,a} dx_a (y_a - \bar{y}) + B - d\bar{x} \left( \sum_{i \in I_{xy}} (w_{xy,i} y_i) - \bar{y} \sum_{i \in I_{xy}} w_{xy,i} \right) \end{aligned}$$

$$\begin{aligned}
C' &= \sum_{i \in I_{xy}} w'_{x,i} (x'_i - \bar{x}')^2 \\
&= w'_{x,a} (x'_a - \bar{x}')^2 + \sum_{i \in I'_{xy}} w_{x,i} (x_i - \bar{x}')^2 \\
&= (w_{x,a} + dw_{x,a}) (x'_a - \bar{x}')^2 + \sum_{i \in I'_{xy}} w_{x,i} (x_i - \bar{x}')^2 \\
&= (w_{x,a} + dw_{x,a}) (x_a - \bar{x}' + dx_a)^2 + \sum_{i \in I'_{xy}} w_{x,i} (x_i - \bar{x}')^2 \\
&= dw_{x,a} (x'_a - \bar{x}')^2 + w_{x,a} (x_a - \bar{x}')^2 + 2w_{x,a} dx_a (x_a - \bar{x}') + w_{x,a} dx_a^2 \\
&\quad + \sum_{i \in I'_{xy}} w_{x,i} (x_i - \bar{x}')^2 \\
&= dw_{x,a} (x'_a - \bar{x}')^2 + 2w_{x,a} dx_a (x_a - \bar{x}') + w_{x,a} dx_a^2 + \sum_{i \in I_{xy}} w_{x,i} (x_i - \bar{x}')^2 \\
&= dw_{x,a} (x'_a - \bar{x}')^2 + 2w_{x,a} dx_a (x_a - \bar{x}') + w_{x,a} dx_a^2 + C \\
&\quad - 2d\bar{x} \left( \sum_{i \in I_{xy}} w_{x,i} x_i - \bar{x} \sum_{i \in I_{xy}} w_{x,i} \right) + d\bar{x}^2 \sum_{i \in I_{xy}} w_{x,i}
\end{aligned}$$

$$\begin{aligned}
D' &= \sum_{i \in I_{xy}} w'_{y,i} (y_i - \bar{y})^2 = w'_{y,a} (y_i - \bar{y})^2 + \sum_{i \in I'_{xy}} w_{y,i} (y_i - \bar{y})^2 \\
&= (w_{y,a} + dw_{y,a}) (y_i - \bar{y})^2 + \sum_{i \in I'_{xy}} w_{y,i} (y_i - \bar{y})^2 \\
&= dw_{y,a} (y_i - \bar{y})^2 + w_{y,a} (y_i - \bar{y})^2 + \sum_{i \in I'_{xy}} w_{y,i} (y_i - \bar{y})^2 \\
&= dw_{y,a} (y_i - \bar{y})^2 + D
\end{aligned}$$

## Appendix C

### Feature types of the evaluation

Altered Vocal Sound  
Blues Influences  
Celtic Influences  
Classical Influences  
Country Influences  
Disco Influences  
Female Vocal  
Flute Solo  
Folk Influences  
Funk Influences  
Group Vocals  
Guitar  
Highly Repetitive Melody  
High-Pitched Voice  
Hip Hop Influences  
House Roots  
Indian Influences  
Jazz Influences  
Latin Influences  
Lyrics  
Lyric-Centric Composition

Male Vocal  
Multiple Vocalists  
Organ  
Piano  
Piano Solo  
Psychedelic Influences  
Punk Influences  
Rap Influences  
R&B Influences  
Reggae Influences  
Repetitive Song Structure  
Rock Influences  
Sax Strings  
Strong Melodies  
Synth  
Techno Roots  
Trance Roots  
Trip Hop Roots  
Trumpet Solo  
Violin Solo

# Bibliography

- Adams, B. D. (2005). Trust vs. confidence. Technical report, Defence Research and Development Canada.
- Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749.
- Anscombe, F. J. (1973). Graphs in statistical analysis. *The American Statistician*, 27(1):17–21.
- Balabanović, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72.
- Carroll, J. (1961). The nature of the data, or how to choose a correlation coefficient. *Psychometrika*, 26(4):347–372.
- Gutscher, A., Heesen, J., and Siemoneit, O. (2008). Possibilities and limitations of modeling trust and reputation. In Möller, M., Roth-Berghofer, T., and Neuser, W., editors, *WSPI*, volume 332 of *CEUR Workshop Proceedings*, pages 50–61. CEUR-WS.org.
- Jøsang, A., Ismail, R., and Boyd, C. (2007). A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644.
- Kiesel, M. and Schwarz, S. (2008). Skipforward - a lightweight ontology-based peer-to-peer recommendation system. In Bizer, C. and Joshi, A., editors, *International Semantic Web Conference (Posters & Demos)*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- Linden, G., Smith, B., and York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, pages 76–80.
- Liu, Z., Wang, Y., and Chen, T. (1998). Audio feature extraction and analysis for scene segmentation and classification. *The Journal of VLSI Signal Processing*, 20(1):61–79.

- Marsh, S. P., of Computing Science, D., Mathematics, and of Stirling, U. (1995). *Formalising Trust as a Computational Concept*. Dept. of Computing Science and Mathematics, University of Stirling.
- Massa, P. and Avesani, P. (2004). Trust-aware collaborative filtering for recommender systems. *Lecture Notes in Computer Science*, pages 492–508.
- Massa, P. and Bhattacharjee, B. (2004). Using trust in recommender systems: An experimental analysis. *Lecture Notes in Computer Science*, pages 221–235.
- Maurer, U. (1996). Modelling a public-key infrastructure. *Lecture Notes in Computer Science*, pages 325–350.
- O’Donovan, J. and Smyth, B. (2005). Trust in recommender systems.
- O’Mahony, M. P., Hurley, N. J., and Silvestre, G. C. M. (2002). Promoting recommendations: An attack on collaborative filtering. In *Proceedings of the 13th International Conference on Database and Expert Systems Applications (DEXA), 2nd–6th*, volume LNCS 2453, pages 494–503, Aix-en-Provence, France. Springer-Verlag.
- Papagelis, M., Rousidis, I., Plexousakis, D., and Theoharopoulos, E. (2005). Incremental collaborative filtering for highly-scalable recommendation algorithms. In Hacid, M.-S., Murray, N. V., Ras, Z. W., and Tsumoto, S., editors, *ISMIS*, volume 3488 of *Lecture Notes in Computer Science*, pages 553–561. Springer.
- Pazzani, M. (1999). A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5):393–408.
- Rodgers, J. L. and Nicewander, A. W. (1988). Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66.
- Sarwar, B., Karypis, G., Konstan, J., and Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *WWW ’01: Proceedings of the 10th international conference on World Wide Web*, pages 285–295, New York, NY, USA. ACM.
- Sarwar, B., Karypis, G., Konstan, J., Riedl, J., and SCIENCE, M. U. M. D. O. C. (2000). *Application of Dimensionality Reduction in Recommender System-A Case Study*. Defense Technical Information Center.
- Schafer, J. B., Konstan, J., and Riedi, J. (1999). Recommender systems in e-commerce. In *EC ’99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166, New York, NY, USA. ACM.
- Shafer, G. (1976). *A mathematical theory of evidence*. Princeton University Press Princeton, NJ.

- Shardanand, U. and Maes, P. (1995). Social information filtering: algorithms for automating "word of mouth". In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.
- Wang, Y. and Vassileva, J. (2003). Trust and reputation model in peer-to-peer networks. *Peer-to-Peer Computing, IEEE International Conference on*, pages 150–157.
- Ziegler, C. and Lausen, G. (2004). Analyzing correlation between trust and user similarity in online communities. *Lecture Notes in Computer Science*, pages 251–265.