RECOMMENDATION IN ENTERPRISE 2.0 SOCIAL MEDIA STREAMS

Dissertation

for obtaining the academic degree of
DOKTOR-INGENIEUR (DR.-ING.)

at
Technische Universität Dresden
Faculty of Computer Science

by
Dipl.-Inf. Torsten Lunze
born December 07, 1979 in Radebeul

Reviewers:
Professor Dr. rer. nat. habil. Dr. h. c. Alexander Schill, TU Dresden
Professor Dr. rer. nat. Florian Matthes, TU München

Day of Defence: September 17, 2014

Dresden, September, 2014
Statement of Authorship

This dissertation has been conducted and presented solely by myself. I have not made use of other people’s work (published or otherwise) or presented it here without acknowledging the source of all such work.

Dresden, September 29, 2014

Torsten Lunze
Acknowledgement

This dissertation was conducted as part of two joint research projects between Communote GmbH and Chair of Computer Networks at Technische Universität Dresden. Therefore I want to thank Dirk Röhrborn (CEO of Communote GmbH) for initiating those research projects and I also want to thank Prof. Alexander Schill for supervising my dissertation and providing me with regular and valuable feedback.

Additionally, I want to thank Dr. Marius Feldmann and Philipp Katz for taking their time to discuss and review my work. I also want to thank Amy Hurtado and my sister Dr. Anja Lunze for proofreading my dissertation back and forth. Last but not least I want to thank my family and friends for supporting me through the times of hard work.
Abstract

A social media stream allows users to share user-generated content as well as aggregate different external sources into one single stream. In Enterprise 2.0 such social media streams empower co-workers to share their information and to work efficiently and effectively together while replacing email communication. As more users share information it becomes impossible to read the complete stream leading to an information overload. Therefore, it is crucial to provide the users a personalized stream that suggests important and unread messages. The main characteristic of an Enterprise 2.0 social media stream is that co-workers work together on projects represented by topics: the stream is topic-centered and not user-centered as in public streams such as Facebook or Twitter.

A lot of work has been done dealing with recommendation in a stream or for news recommendation. However, none of the current research approaches deal with the characteristics of an Enterprise 2.0 social media stream to recommend messages. The existing systems described in the research mainly deal with news recommendation for public streams and lack the applicability for Enterprise 2.0 social media streams.

In this thesis a recommender concept is developed that allows the recommendation of messages in an Enterprise 2.0 social media stream. The basic idea is to extract features from a new message and use those features to compute a relevance score for a user. Additionally, those features are used to learn a user model and then use the user model for scoring new messages. This idea works without using explicit user feedback and assures a high user acceptance because no intense rating of messages is necessary. With this idea a content-based and collaborative-based approach is developed. To reflect the topic-centered streams a topic-specific user model is introduced which learns a user model independently for each topic.

There are constantly new terms that occur in the stream of messages. For improving the quality of the recommendation (by finding more relevant messages) the recommender should be able to handle the new terms. Therefore, an approach is developed which adapts a user model if unknown terms occur by using terms of similar users or topics. Also, a short- and long-term approach is developed which tries to detect short-term interests
of users. Only if the interest of a user occurs repeatedly over a certain time span are terms transferred to the long-term user model.

The approaches are evaluated against a dataset obtained through an Enterprise 2.0 social media stream application. The evaluation shows the overall applicability of the concept. Specifically the evaluation shows that a topic-specific user model outperforms a global user model and also that adapting the user model according to similar users leads to an increase in the quality of the recommendation. Interestingly, the collaborative-based approach cannot reach the quality of the content-based approach.
2.9.4 Applicable on Enterprise 2.0 Streams ............................. 19
2.9.5 Be Adaptive ......................................................... 19
2.9.6 Incremental Integration .......................................... 19
2.9.7 Recommendation in near Real-Time .......................... 19
2.9.8 Quality of the Recommendation Algorithm .................. 19
2.9.9 Non-Requirements ............................................. 19

3 Research Approach .................................................. 21
  3.1 Research Challenges ............................................ 21
  3.2 Approach .......................................................... 22
    3.2.1 Topic-Specific Enterprise 2.0 Social Media Stream .......... 22
    3.2.2 Adapt to new Terms ........................................ 22
    3.2.3 Short-Term and Long-Term Interests ........................ 23
  3.3 Research Questions and Theses .................................. 24
  3.4 Out of Scope ..................................................... 25

4 State of the Art ...................................................... 27
  4.1 Recommender Types .............................................. 27
    4.1.1 Classification ............................................... 27
    4.1.2 Content-Based Recommender ................................ 29
    4.1.3 Collaborative-Based Recommender ......................... 30
    4.1.4 Community-based Recommender ............................ 31
    4.1.5 Comparing Recommender Types against Requirements .... 31
  4.2 Typical Recommender Architecture ............................ 33
  4.3 Information Extraction for Recommender ....................... 35
  4.4 User Models ..................................................... 35
    4.4.1 Representation of User Models ............................. 36
    4.4.2 Profile Generation ......................................... 37
    4.4.3 Learning of User Models ................................... 37
    4.4.4 Adaptation of User Models ................................. 37
  4.5 Concept Drift and Shift ......................................... 38
  4.6 Existing Relevant Systems ....................................... 39
    4.6.1 IBM Connections ............................................ 40
    4.6.2 Google News ................................................ 40
    4.6.3 ALOE, OrganiK: Recommendation of Latent Topics .......... 41
    4.6.4 SCENE, PENETRATE: Personalized News Recom-
        mendation .................................................... 42
    4.6.5 Tag-Based Recommender .................................... 43
    4.6.6 Yahoo News Recommender ................................ 43
    4.6.7 Hermes: Semantic-Based Approach .......................... 44
    4.6.8 Twitter Recommenders ...................................... 44
    4.6.9 Facebook ..................................................... 44
    4.6.10 Comparison of Existing Systems ............................ 46
4.7 Limitations of State of the Art Approaches for Enterprise 2.0
Social Media Stream Recommender ........................................ 47

5 Recommender Concept ..................................................... 51
5.1 Basic Architecture ..................................................... 51
5.1.1 Interfaces .................................................................. 51
5.1.2 Dependent Components ........................................... 53
5.1.3 Recommendation Subcomponents ............................... 54
5.1.4 Architectural Requirements ....................................... 54
5.2 Basic Algorithm .......................................................... 55
5.2.1 Features .................................................................. 56
5.2.2 Recommender Model ................................................ 56
5.3 Recommendation Process .............................................. 59
5.3.1 Scoring Process ....................................................... 59
5.3.2 Learning Process ..................................................... 62
5.4 Content-Based Approach .............................................. 63
5.4.1 Learning User Models ............................................... 64
5.4.2 Term Weights ........................................................ 65
5.4.3 Content Match Feature ............................................ 66
5.4.4 Cold Start .............................................................. 66
5.5 User Model Adaptation Approach ................................. 68
5.5.1 User Model Adaptation by Similar Users ................. 69
5.5.2 User Model Adaptation by Similar Topics ................ 72
5.5.3 User Model Adaptation by Similar Users and Topics .... 73
5.5.4 Adaptation and Learning .......................................... 74
5.5.5 Change of Similarity over Time ................................. 74
5.6 Short-Term and Long-Term Approach ......................... 74
5.6.1 Time Decay .......................................................... 75
5.6.2 Long-Term User Model ............................................ 76
5.7 Collaborative-Based Approach .................................... 78
5.7.1 Collaboration on Ratings .......................................... 79
5.7.2 Collaboration on Observations ................................. 79
5.7.3 Collaboration on Terms ........................................... 80
5.8 Genetic Algorithm for Optimization ............................. 81
5.9 Compare Approach to Requirements ......................... 83

6 Realization ................................................................. 85
6.1 SPEKTRUM .............................................................. 85
6.1.1 Components ........................................................ 86
6.1.2 Data Model .......................................................... 86
6.1.3 Communicator ....................................................... 88
6.1.4 Persistence ............................................................ 88
6.2 SPEKTRUM Intelligence ............................................. 90
6.2.1 Configuration and Initialization ............................... 90
## Contents

6.2.2 Scoring Process ................................................. 90  
6.2.3 Learning Process ............................................. 92  
6.3 Integration of SPEKTRUM into Applications ................ 93  
6.4 Communote ......................................................... 94  
6.4.1 Communote Architecture ................................. 94  
6.4.2 Communote Integration ................................. 94  
6.4.3 Communote Data Model ...................................... 95  
6.4.4 MyStream Communote Plugin ......................... 96  
6.4.5 Front-End and Interaction .............................. 96  
6.4.6 Initial User Model Learning ......................... 97  

7 Evaluation ............................................................. 99  
7.1 Evaluation and Requirements ............................... 99  
7.2 Dataset for Evaluation ........................................ 101  
7.3 Measures for Evaluation ...................................... 107  
7.3.1 Basic Definitions ....................................... 108  
7.3.2 Basic Measures .......................................... 109  
7.3.3 Time-Binned Measures .................................. 112  
7.4 Implementation of Evaluation ............................... 114  
7.5 Basic Algorithm Optimization ............................. 114  
7.5.1 Configuration ............................................ 114  
7.5.2 Baseline .................................................... 117  
7.6 Scoring Feature Optimization ............................... 118  
7.7 Evaluation of Content-Based Approach ................. 119  
7.7.1 Optimization ............................................ 119  
7.7.2 Results .................................................. 120  
7.7.3 Interpretation .......................................... 122  
7.8 Evaluation of User Model Adaptation Approach .... 124  
7.8.1 Optimization ............................................ 124  
7.8.2 Results .................................................. 124  
7.8.3 Interpretation .......................................... 126  
7.9 Evaluation of Short-Term and Long-Term Approach .... 129  
7.9.1 Optimization ............................................ 129  
7.9.2 Results .................................................. 130  
7.9.3 Interpretation .......................................... 131  
7.10 Evaluation of Collaboration-Based Approach ........ 132  
7.10.1 Optimization ........................................... 133  
7.10.2 Results .................................................. 133  
7.10.3 Interpretation .......................................... 133  
7.11 Comparison of Different Approaches .......... 135  

xii
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Example Twitter message stream.</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Example Facebook message stream.</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Communote, an Enterprise 2.0 Social Media Stream Application.</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Yammer, an Enterprise 2.0 Social Media Stream Application.</td>
<td>11</td>
</tr>
<tr>
<td>2.5</td>
<td>The model of a typical Enterprise 2.0 Social Media Stream Application.</td>
<td>12</td>
</tr>
<tr>
<td>2.6</td>
<td>For the problem of information overload the solution is information filtering. (Both figures taken from [LF1I].)</td>
<td>13</td>
</tr>
<tr>
<td>2.7</td>
<td>Scenario: Changing interest in a feature discussion.</td>
<td>15</td>
</tr>
<tr>
<td>2.8</td>
<td>Scenario: Changing interest in a new technology discussion.</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>Overlapping terms in topics (projects).</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Typical recommender architecture.</td>
<td>34</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview of recommendation architecture.</td>
<td>52</td>
</tr>
<tr>
<td>5.2</td>
<td>Learn from features.</td>
<td>56</td>
</tr>
<tr>
<td>5.3</td>
<td>Detailed recommendation process.</td>
<td>60</td>
</tr>
<tr>
<td>5.4</td>
<td>Example for a user model and for a computation of the Content Match Feature (CMF).</td>
<td>67</td>
</tr>
<tr>
<td>5.5</td>
<td>Example for applying User Model Adaptation (UMA).</td>
<td>70</td>
</tr>
<tr>
<td>5.6</td>
<td>Separate messages in time bins.</td>
<td>75</td>
</tr>
<tr>
<td>5.7</td>
<td>Separating term occurrence in time bins.</td>
<td>75</td>
</tr>
<tr>
<td>5.8</td>
<td>Periodic occurrences of interest terms in a time-binned user model.</td>
<td>77</td>
</tr>
<tr>
<td>5.9</td>
<td>Permanent occurrences of interest terms in a time-binned user model.</td>
<td>77</td>
</tr>
<tr>
<td>5.10</td>
<td>User to Message (UtM) collaboration-based model.</td>
<td>79</td>
</tr>
<tr>
<td>5.11</td>
<td>User To Term (UtT) collaboration-based model.</td>
<td>80</td>
</tr>
<tr>
<td>5.12</td>
<td>User To Term (UtT) collaboration-based model per topic.</td>
<td>81</td>
</tr>
<tr>
<td>5.13</td>
<td>Example for a Genetic Algorithm: Representation of feature values as gen including an example for crossover and mutation.</td>
<td>82</td>
</tr>
</tbody>
</table>
### List of Figures

6.1 This UML diagram shows the adapter architecture of an Open Source Framework for Social Media Stream Recommendation (SPEKTRUM) component with relevant implementations. 87

6.2 This UML class diagram shows the core data elements of the SPEKTRUM component. 89

6.3 This UML component diagram shows the main components and their dependencies. 95

6.4 Communote rating front-end implementation. 96

7.1 Rating Sets. 101

7.2 Number of ratings for each user with absolute values. 103

7.3 Rating distribution per user for each month. 104

7.4 Rating distribution per user for Ratings with Interaction (RwI) and Ratings without Interaction (RwOI). 105

7.5 Number of ratings per day for all messages. 106

7.6 Number of Ratings with Interaction (RwI) for each day. 106

7.7 Number of Ratings without Interaction (RwOI) for each day. 107

7.8 UML class diagram of the evaluation implementation. 115

7.9 Evaluation results for feature- and content-based runs for Ratings without Interaction (RwOI). 122

7.10 Evaluation results for feature- and content-based runs for Ratings without Discussion Interaction (RwoDI). 123

7.11 Evaluation results for UMA variants for RwOI. 128

7.12 Comparing different approaches for All Ratings (Rall). 138

7.13 Comparing different approaches for Ratings without Interaction (RwOI). 139

7.14 Comparing different approaches for Ratings without Discussion Interaction (RwoDI). 140

A.1 Self-Rating Evaluation: Distribution per importance. 151

A.2 Self-Rating Evaluation: Distribution per importance and urgency. 152

A.3 Self-Rating Evaluation: Term change compared to adjacent month. 155

A.4 Self-Rating Evaluation: User model term change compared to adjacent month. 157

A.5 Self-Rating Evaluation: User model term change compared to previous bin of length month and each bin moved by one day. 158

A.6 Self-Rating Evaluation: User model term change compared to previous bin of length month and each bin moved by one week. 158
# List of Tables

<table>
<thead>
<tr>
<th>Table No.</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Comparing recommender types against requirements</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of the state of the art existing recommender systems</td>
<td>49</td>
</tr>
<tr>
<td>5.1</td>
<td>Message features in the scoring process</td>
<td>57</td>
</tr>
<tr>
<td>5.2</td>
<td>Example for features in the scoring process for two discussions</td>
<td>58</td>
</tr>
<tr>
<td>5.3</td>
<td>Comparison of concept to requirements</td>
<td>84</td>
</tr>
<tr>
<td>7.1</td>
<td>Number of positive (relevant), negative (irrelevant) and all ratings</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>for each rating set of the evaluation dataset</td>
<td></td>
</tr>
<tr>
<td>7.2</td>
<td>Configuration options for information extraction</td>
<td>116</td>
</tr>
<tr>
<td>7.3</td>
<td>Configuration options for the Content Match Feature (CMF)</td>
<td>116</td>
</tr>
<tr>
<td>7.4</td>
<td>Optimized configuration for the information extraction and</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>Content Match Feature (CMF)</td>
<td></td>
</tr>
<tr>
<td>7.5</td>
<td>Optimized configuration for the scoring feature weights including CMF and</td>
<td>119</td>
</tr>
<tr>
<td></td>
<td>Collaboration Match Feature (CbMF)</td>
<td></td>
</tr>
<tr>
<td>7.6</td>
<td>Optimized configuration for learning a user model</td>
<td>120</td>
</tr>
<tr>
<td>7.7</td>
<td>Results for content-based approaches for All Ratings (Rall)</td>
<td>121</td>
</tr>
<tr>
<td>7.8</td>
<td>Results for content-based approaches for Ratings without</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interaction (RwoI)</td>
<td>121</td>
</tr>
<tr>
<td>7.9</td>
<td>Results for content-based approaches for Ratings without</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Discussion Interaction (RwoDI)</td>
<td>122</td>
</tr>
<tr>
<td>7.10</td>
<td>Optimized configuration for UMA</td>
<td>125</td>
</tr>
<tr>
<td>7.11</td>
<td>Results for UMA approaches for All Ratings (Rall)</td>
<td>126</td>
</tr>
<tr>
<td>7.12</td>
<td>Results for UMA approaches for Ratings without Interaction (RwoI)</td>
<td>127</td>
</tr>
<tr>
<td>7.13</td>
<td>Results for UMA approaches for Ratings without Discussion Interaction (RwoDI)</td>
<td>127</td>
</tr>
<tr>
<td>7.14</td>
<td>Results for UMA approaches using User-Model-Based Similarity (UMS)</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>with different top-n user configurations for All Ratings (Rall)</td>
<td></td>
</tr>
</tbody>
</table>
List of Tables

7.15 Results for UMA approaches using User-Model-Based Similarity (UMS) with different top-n user configurations for Ratings without Interaction (RwoI) .................................. 129
7.16 Optimized configuration for short-term approach. .............. 130
7.17 Results for short-term approaches for All Ratings (Rall) ........ 131
7.18 Results for short-term approaches for Ratings without Interaction (RwoI) ........................................ 131
7.19 Results for short-term approaches for Ratings without Discussion Interaction (RwoDI) ........................................ 132
7.20 Results for collaboration-based approaches for All Ratings (Rall) .................................................. 133
7.21 Results for collaboration-based approaches for Ratings without Interaction (RwoI) ........................................ 134
7.22 Results for collaboration-based approaches for Ratings without Discussion Interaction (RwoDI) ........................................ 134
7.23 Results for various top time-binned measures for best of different approaches for All Ratings (Rall) .................. 136
7.24 Results for various top time-binned measures for best of different approaches for Ratings without Interaction (RwoI) .. 136
7.25 Results for various top time-binned measures for best of different approaches for Ratings without Discussion Interaction (RwoDI) ........................................ 137

A.1 Statistics for the features of the self-evaluation pre-analysis. 153
A.2 Dataset statistics for the discussion-based generated ratings. 156
A.3 Mapping of importance to rating values. .......................... 156
Abbreviations

$AP@k$  Average-Precision@$k$
ASF  Atom Syndication Format
CB  Content-Based
CbB  Collaboration-Based
CbMF  Collaboration Match Feature
CMF  Content Match Feature
CS  Combined Similarity using Topics and Users
E2S  Enterprise 2.0 Social Media Stream
E2SA  Enterprise 2.0 Social Media Stream Application
E2SR  Enterprise 2.0 Social Media Stream Recommender
$F_1$  $F_1$-Score
$F_2$  $F_2$-Score
GA  Genetic Algorithm
$imf$  Inverse Message Frequency
JDBC  Java Database Connectivity
JMS  Java Messaging Service
JPA  Java Persistence API
JVM  Java Virtual Machine
HTML  Hypertext Markup Language
LDA  Latent Dirichlet Allocation
LSH  Locality Sensitive Hashing
LT  Long-Term
LTR  Learning to Rank
$MAP$  Mean-Average-Precision
NE  Named Entities
NER  Named Entity Recognition
NLP  Natural Language Processing
<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoSQL</td>
<td>Not only SQL</td>
</tr>
<tr>
<td>OSGI</td>
<td>Open Services Gateway initiative</td>
</tr>
<tr>
<td>$P$</td>
<td>Precision</td>
</tr>
<tr>
<td>$P@k$</td>
<td>Precision@$k$</td>
</tr>
<tr>
<td>PLSI</td>
<td>Probabilistic Latent Semantic Indexing</td>
</tr>
<tr>
<td>Rall</td>
<td>All Ratings</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>$R$</td>
<td>Recall</td>
</tr>
<tr>
<td>RSS</td>
<td>Really Simple Syndication</td>
</tr>
<tr>
<td>RwDI</td>
<td>Ratings with Discussion Interaction</td>
</tr>
<tr>
<td>RwI</td>
<td>Ratings with Interaction</td>
</tr>
<tr>
<td>RwoDI</td>
<td>Ratings without Discussion Interaction</td>
</tr>
<tr>
<td>RwoI</td>
<td>Ratings without Interaction</td>
</tr>
<tr>
<td>SPEKTRUM</td>
<td>Open Source Framework for Social Media Stream Recommendation</td>
</tr>
<tr>
<td>ST</td>
<td>Short-Term</td>
</tr>
<tr>
<td>$tf - idf$</td>
<td>Term Frequency - Inverse Document Frequency</td>
</tr>
<tr>
<td>TB</td>
<td>Time-Binned</td>
</tr>
<tr>
<td>$TB-MAP$</td>
<td>Time-Binned Mean-Average-Precision</td>
</tr>
<tr>
<td>$TB-P@k$</td>
<td>Time-Binned Precision@$k$</td>
</tr>
<tr>
<td>TS</td>
<td>Topic Similarity</td>
</tr>
<tr>
<td>UM</td>
<td>User Model</td>
</tr>
<tr>
<td>UMA</td>
<td>User Model Adaptation</td>
</tr>
<tr>
<td>UMS</td>
<td>User-Model-Based Similarity</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>UtM</td>
<td>User to Message</td>
</tr>
<tr>
<td>UtT</td>
<td>User To Term</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XMPP</td>
<td>Extensible Messaging and Presence Protocol</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

As the internet has grown over the past few decades, web services have sprung up and begun blending together. In this last decade, content publishing has intertwined with collaboration-based services. People interact with each other using social networks such as Facebook or Twitter.

This development continues to grow into enterprises. Here, wikis, collaboration tools and social media suites have been created in the past few years and are being deployed in enterprises, commonly known as Enterprise 2.0.

This progress leads to more and more information for the knowledge worker to keep up with: new technologies arise, new discussions evolve and in order to stay ahead of competitors, the knowledge worker not only needs relevant information but he or she also needs to get relevant information fast. Therefore, a personalization of the information stream is necessary to support information selection.

1.1 Focus

This thesis focuses on recommendation algorithms that can be applied to a social media stream of an Enterprise 2.0 Social Media Stream Application (E2SA). Such a recommendation algorithm should help the user to identify messages relevant to his or her tasks. The recommendation algorithm must fulfill several requirements such as working without explicit user feedback, working on short unstructured texts and dealing with changing interests of users.

1.2 Research Approach

The main goal of this thesis is to increase the quality of recommendation for an Enterprise 2.0 Social Media Stream Recommender (E2SR). The quality
can be measured by different scores. The main measures used in this thesis are the $F$-Scores and Time-Binned Mean-Average-Precision (TB-MAP). The $F$-Scores measure the quality based on correct and precise recommendations over the whole dataset. The TB-MAP is based on the top-$n$ messages of each time bin (e.g. day or week). The goal of this thesis is to research three questions:

1. **Recommendation in Enterprise 2.0 Streams**: How can the quality of recommendation (measured by $F$-Scores and TB-MAP) be improved in an E2SR without using explicit user feedback? This question focuses on a basic recommender that fulfills the relevant requirements such that the recommender can be easily integrated into a productive E2SA while tackling the Enterprise 2.0 Social Media Stream (E2S) characteristics. The main goals here are to find a way to learn from features that can be determined based on the user and message structure and that reflect the topic-specific message structure of an E2S.

2. **Adapt to new interests**: How can the quality of recommendation (measured by $F$-Scores and TB-MAP) be improved by adapting user models to new interests in an E2SR? The content of messages changes and new topics arise. To improve the quality of the recommendation, the idea is to treat new unknown content of messages differently. For example, if a recommendation for a user lacks certainty the recommendation may use the information from similar users to make a recommendation.

3. **Short-Term and Long-Term Interests**: How can a separation of short- and long-term interests improve the quality of recommendation (measured by $F$-Scores and TB-MAP) in an E2SR? Besides new interests, the user might also only be interested in a topic for a short period of time. This requires firstly identifying short-term interests of a user, and then secondly using them to recommend messages that fit the short-term interests as they exist.

Based on those questions the following theses are claimed:

1. A content-based recommender with a topic-specific user model will lead to better recommendation quality than one with a global user model.

2. Adapting unknown terms in the user model by similar users or similar topics will lead to an increase in the quality of recommendation.

3. Separating short- and long-term interest will lead to an increase in the recommendation quality.
1.3 Scope

In the current state of the art a lot of research on different types of recommenders has been done. A part of this research deals with news recommendation or with recommendation in social media sharing platforms. Only [Guy+10; GRR11] deals with recommendation within enterprise applications, but none of the existing approaches considers recommendation in an E2S in detail.

[Guy+10; GRR11] mainly exploits the user network structure to make recommendations. This approach cannot be directly used for a typical E2S since the user network structure is not available, as in the scenario of [GRR11]. Other approaches which deal with message streams are based on Twitter data and are only applicable to specific Twitter behavior (intensive use of retweet and hash-tags by large user groups). Twitter approaches mainly focus on detection of emerging topics [CDCS10; Che+13], detecting users to follow [HBS10] or using the retweets to determine popularity [HDD11; Nav+11; Alh+12]. Also, the structure of Twitter and its data is different from the typical structure of an E2SA.

In [Das+07; Li+11b; Zhe+13] recommenders for news platforms are described. Those approaches are specialized to handle an extreme large amount of news and are optimized to pick the most relevant message for a user based on a click history. None of them handle the characteristics of an E2S. In particular, when the interactions for a news message are very rare those algorithms will not be able to recommend such news message. However, for an E2SA it is crucial to also recommend messages with few to zero interactions.

The approaches described in [Sch+11; CMA12] are applied within an enterprise application but they do not recommend messages. Instead, they recommend latent topics or resources based on a selected tag or resource. They lack the ability to recommend messages per user.

None of the existing approaches is directly applicable to recommend messages in E2S and all existing systems lack the ability to be applied to an E2SA. Basic algorithms and methods can be used but must be assembled in a novel approach to match the requirements for recommendation in an E2S.

1.4 Overview

In Chapter 2, the problem of recommendation in an E2S is analyzed. E2SA will be introduced and examples will be given. What will be discussed is what makes E2SA different from other stream applications. Based on the scenarios, use case and requirements will be defined.
1 Introduction

In Chapter 3, the research approach will be formulated. The research questions will be introduced and discussed. Based on the questions, three theses will be claimed.

Focusing on the problem and the research question, the state of the art will be discussed in Chapter 4. First, different recommender types will be discussed as well as related concepts relevant to this thesis such as concept drift and shift. Then, existing systems that have similarity to the analyzed problem will be described. The chapter will close with an analysis of the limitations of the State of the Art E2SR.

The detailed concept of how to solve the research questions will be discussed in Chapter 5. The basic system architecture will be described. The following will be discussed: how an E2SR can learn without explicit feedback, how user model adaptation can be used and how short-term interests can be determined. The detailed concept will include several approaches to solve the requirements: a content-based, a user model adaptation and a short-term approach. Also a collaborative approach is developed for comparison.

The algorithms of the concept have been implemented in an open source framework named SPEKTRUM. Details of the implementation and the configuration of the algorithm will be presented in Chapter 6.

A detailed evaluation of the concept against an E2SA dataset will be shown in Chapter 7. For each research question and claimed thesis, different configurations will be analyzed and compared with each other.

Finally, in Chapter 8, the results of this work will be summarized. This includes a discussion which approach leads to an improvement in recommendation quality of an E2SR and which does not. The research questions will be answered and it will be explained whether each claimed thesis is true or not. Also, future work will be discussed.

In Appendix A, the results of a pre-analysis which inspired the research questions of this thesis are discussed. The pre-analysis was conducted using a E2SA dataset and investigated the behavior of change over time in terms and interests of messages.
Chapter 2

Problem Analysis

The goal of this chapter is to provide a detailed description of the problem to be solved. First is a description of what Social Media Streams (Section 2.1) are and what Enterprise 2.0 (Section 2.2) is. Then, Enterprise 2.0 Social Media Stream (E2S) are introduced (Section 2.3) and examples for Enterprise 2.0 Social Media Stream Application (E2SA) are given (Section 2.4). The necessities for recommendation within streams are described in Section 2.5. In Section 2.6 the characteristics of E2SA are discussed.

Next, in Section 2.7 scenarios for recommendation in an E2SA are described in detail and use cases for recommendation are defined in Section 2.8. Also, requirements are formulated in Section 2.9 that will later be used for the comparison against the state of the art (Chapter 4) and the concept (Chapter 5).

2.1 Social Media Streams

In the first stage of the Web 2.0, users started to generate content in an intense manner using wikis and blogs. Later, social networking sites such as LinkedIn, Xing (both founded 2003) and Facebook (founded 2004) became popular that allow users to connect to each other, share status updates and exchange messages. The microblogging service Twitter was founded 2006. Twitter allows users to share text messages limited to 140 characters.

Of course, all those platforms have been developed through the last years and integrated more features with focus on their specific usage. LinkedIn and Xing focus on business connections, Facebook on personal connections and Twitter as a mix of everything.

All those platforms more or less include social media streams. The stream consists of items or messages which represent (but are not limited to) one of the following:

- text messages
2 Problem Analysis

- videos or links to videos
- pictures
- polls
- new connections between users

Depending on the system for each item or message, users can interact with the item to like, share, favor or comment the message.

In the stream, the messages are typically ordered by time showing the most recent message first. Some applications, e.g. Facebook, have some form of alternative ordering, showing important items on top. Typically, the older a message is, the less likely it will be on top. For example Twitter and Facebook streams are shown in Figure 2.1 and 2.2, respectively.

![Twitter Message Stream](image)

Figure 2.1: Example Twitter message stream.

**Activity Streams**

A special form of social message stream is an Activity Stream \([\text{Act12a}]\) which focuses on the activities within an application. An activity typically has the form User X commented on the task 'ABC' consisting of:

**Actor** The person (or a system) triggering the activity.

**Verb** The action taken place, e.g. comment, upload, like.
2.1 Social Media Streams

Im Institut für Algebra der TU Dresden ist in dieser Woche ein neuer Weltrekord in der Kryptographie aufgestellt worden. Der Marie-Curie-Siopendiat Dr. Jens Zumbrägel führte eine Attacke auf das sogenannte diskrete Logarithmusproblem durch, das die Grundlage für viele wichtige Arten moderner Verschlüsselungsverfahren ist, die beispielsweise beim E-Banking eingesetzt werden. Zumbrägel berechnete mit einem internationalen Forscherteam unter Verwendung eines Hochleistungsknoten einen diskreten Logarithmus in einem endlichen Körper der Größe 2 hoch 9234 (29234).

Figure 2.2: Example Facebook message stream.
2 Problem Analysis

Object  The object that is part of the activity, e.g. the picture that has been uploaded or the message a comment has been added to.

2.2 Enterprise 2.0

The Web 2.0 also managed its way into the enterprises. Here, the term Enterprise 2.0 refers to the usage of social software within the enterprise for knowledge management, internal and external communication. Andrew McAfee defines in his blog Enterprise 2.0 as:

Enterprise 2.0 is the use of emergent social software platforms within companies, or between companies and their partners or customers.

Social software is used within enterprises to support collaboration between knowledge workers: web-based applications such as Atlassian Confluence or Microsoft SharePoint help the knowledge worker to store, share and manage information, thereby easing collaboration between employees.

2.3 Enterprise 2.0 and Social Media Streams

In the last few years, more applications within the enterprise 2.0 universe have been developed that focus on the internal enterprise communication. Email communication has been replaced by those communication platforms. The advantages of such platforms are transparency, open communication, and easier sharing of information in contrast to email communication. Instead of writing an email, a message is created within the platform and other interactions lead to a stream of messages. In its simplest form, it can be compared to the inbox of an email account.

Besides the social collaboration software tools and the information they are producing, the knowledge worker has to keep up with external information such as Twitter, Facebook and external blogs that may all be relevant for the knowledge worker. Twitter and Facebook are highly relevant for sales, marketing and support requests and external blogs are important for keeping up with a specific domain (e.g. for software engineers developing with Java).

To summarize, the knowledge worker has internal communication and external resources to monitor and to interact with depending on his projects and roles. This is where an E2SA comes in to support the knowledge worker in achieving this.
2.4 Enterprise 2.0 Stream Applications

In this section, two E2SAs are described to get an better understanding of what an E2SA is and how it works.

2.4.1 Communote

Communote[Com14b] is a web-based communication application developed by Communote GmbH[Com14b]. Users within Communote can write and share messages. The main screen of Communote is shown in Figure 2.3. One central aspect of Communote is that messages are organized into topics. Topics can be created by each user and each topic has different rights. The creator or manager of a topic can define which users or user groups can read a topic or write messages into a topic. Those topics are typically used to bundle the communications of a project or a working group.

A message in Communote can be of arbitrary length and can contain simple formats (bold, italic, lists). A message can contain attachments. Pictures and video links will be shown as previews per message. Also tags, either as hash-tags or explicit tags, can be assigned per message.

Within a message users can be mentioned. A mention means that the mentioned user will receive a notification by email or Extensible Messaging
2 Problem Analysis

and Presence Protocol (XMPP) and the message will also appear in the list of all mentions for the user.

Users can reply to a message, thereby forming discussions. The reply can form a thread-like structure of arbitrary depth. Besides that users can like messages or put them on a remember me list. Users can follow other users, topics or tags. Messages that are assigned to a followed topic have a followed tag, or the author is a followed user who will appear in the follow view of the user.

After the login, the user sees all messages he has write-access to, sorted by the creation date of the message with the youngest message first. The user can switch to see only the messages he is mentioned in or only the messages he is following.

The user can filter this list by choosing a topic, authors or tags or by selecting a specific content type (e.g. only show messages with images).

2.4.2 Yammer

Another example of an E2SA is Yammer [Yam14], shown in Figure 2.4. The functionality of Yammer is quite similar to Communote. Here, groups can be formed allowing the restriction of access to a message to only particular users. Also, tags and attachments can be assigned to a new message.

Users can access a list of all messages (called conversations in Yammer) or only messages of a specific group. Also, the stream (or list) can be filtered.

2.4.3 Model of an Enterprise 2.0 Social Media Stream Application

A model with entities and relations of a typical E2SA is defined. This model is used throughout this work for a common understanding. Not all entities or relations will exist in every E2SA but the main aspects will be found in one form or another in each E2SA.

The model is shown in Figure 2.5. The entities of this model are:

User A user within the system can read or write messages. Users can be organized into user groups.

Tag Tags are short strings and are assigned to messages to ease the filtering of the messages with specific tags.

Message A message contains some kind of (media) content and is written by a user within a topic. A message can contain tags.

Topic A topic contains a set of messages and has some kind of defined access rights. There are public topics that are available to all users.

In most E2SA’s a user can follow other users, topics or tags. Typically, the user will then have a view or a filter to show only messages the user follows.

10
2.4 Enterprise 2.0 Stream Applications

Figure 2.4: Yammer, an Enterprise 2.0 Social Media Stream Application.

In messages the author can specify other users who will receive a notification. This is a mention. This way the author indicates that the user should read this message. Sometimes, instead of getting a notification the E2SA provides a view to filter for messages with mentions. Also, the user can like a message to indicate that the content was interesting or to reward the author for sharing the information.

There are different types of access levels that can be granted to a user for a topic. Useful access levels are: Read Access, Write Access and Manage Access. With read access the user can only read messages in the topic. With write access the user can read and write messages in the associated topic. With manage access the user can also change the access rights for the topic and add, change or delete access rights for a user.

A user can reply to messages. The new message is then the reply to the existing one. In some cases the replies can have a thread-like structure. In other cases only one level of replies is allowed, then the replies are sometimes called comments. In all cases the reply structure can form a discussion when several users are involved.
2.5 Recommendation of Messages in Streams

Many messages in one stream will lead to an information overload where the user will be overextended as symbolized in Figure 2.6 on the left side. Within Twitter, reading all messages within of all followers is nearly impossible for most users and not all messages are relevant. Typically, messages (tweets) with a mention or with a lot of retweets are relevant for the user. In an E2S, messages of certain topics are typically more relevant than others. Also, messages in which the user is mentioned will be important for the user.

Besides the easy identification of relevant messages, there is an urge to help the user to find or filter for relevant messages, as shown in Figure 2.6 on the right side. A stream application should therefore have the ability to identify relevant messages for a user. Here, a recommendation algorithm can help to compute a relevance score for a message for a user. Based on that score, a decision can be made whether the message is relevant for the user or not. Also, the score can be used to sort the messages and show the highest scored messages in a list of recommendations.

2.6 Enterprise 2.0 Social Media Stream Characteristics

The difference between the enterprise and the non-enterprise world is fluent. A typical enterprise consists of different departments with different responsibilities but with comprehensive tasks. Within an enterprise there are different projects with employees assigned to those projects. Each employee
2.6 Enterprise 2.0 Social Media Stream Characteristics

Figure 2.6: For the problem of information overload the solution is information filtering. (Both figures taken from [LF11].)

has a certain role with specific responsibilities. Typical roles include project leader, developer, designer and marketing engineer. These roles might be undertaken by multiple persons, or the same person may have different roles in different projects. The tasks within those projects may take days, weeks or years to be completed. Hence, with this dynamic set-up it is clear that the information that is relevant for a single worker is dynamic and changing as well.

Within an enterprise, employees typically form teams of 5-12 persons. The communication within such a team is intense. In contrast, communication on Facebook or Twitter happens in a different form. On Twitter most tweets have a blackboard format, i.e. a user posts something without caring too much who is reading it and who is not. Of course, some tweets have a communication style using mentions or replies, but in an E2S communication is more directed between users as a stream that makes use of the follow principle (as in Twitter).

This is similar to Facebook or LinkedIn. The stream is defined through the respective network of the user. There are options to configure the stream, i.e. by blocking users or by building groups of users. This stream focuses on interesting messages relevant to the user. Within the enterprise - especially in an agile enterprise - the focus and relevance lies also on the team.

As will be described in the state of the art (see Chapter 4), there are a lot of systems dealing with news information. The difference to typical stream messages (or Non-Enterprise 2.0 Streams) is that news articles are (mostly) proofread and optimized (e.g. tags are maintained correctly or optimized for search engines). On the other side, Twitter, Facebook and also enterprise stream messages are often quickly written, not proofread, not well tagged, or never tagged, so it is a more informal writing style.
Another issue in the enterprise is security. The access of users to projects and to information should be regulated. That means personalized algorithms, suggestions and recommendations should not be based on messages the user himself cannot access.

To summarize, the characteristics of an E2S are:

- messages are separated into topics (e.g. projects),
- messages are often very short,
- communication is intense within a team,
- messages are written quickly, often in an informal style.

### 2.7 Scenarios

An IT company COM with different departments and different teams working on different projects and products is assumed. The company uses an E2SA. One team named TIGER uses the Scrum method to develop a web application which allows users to search for and buy chocolate. The team consists of a Scrum Master, a Product Owner, a Software Architect, a Consultant, and a Software Developer.

#### 2.7.1 Good Morning

As one of the first things in the morning the Software Architect takes a look at the mentions and messages from the discussions he is involved in. He replies to those messages insofar as necessary or marks them as further ToDos.

#### 2.7.2 Be Informed

At the end of the day the Software Architect filters for important messages that are not mentions or messages from discussions he is involved in. In this way, a message from another team member shows up that is reporting a technical problem. The Software Architect knows this problem and can reply directly with a message, lending his experience and helping the other team to save time in finding a solution.

#### 2.7.3 Change of Interest

During the product development life cycle, a new feature request comes up and a feature discussion is started by the Product Owner. In the first week, as shown in Figure 2, the product Owner, the Consultant and the Software Architect are discussing the feature to identify different variants and side effects of the feature.
The Product Owner writes a detailed message and mentions the Software Architect and the Consultant. The Consultant makes a phone call to a customer and replies to the message with his status. The Software Architect also thinks about the technical implications of the new feature request and points out problems. The Product Owner - pushed by the stakeholders - tries to force the feature to be implemented immediately. The Scrum Master - not yet mentioned in the discussion - filters for important messages and the
feature request discussion pops up. The Scrum Master joins the discussion and tells the Product Owner to respect the Scrum commitments of the team and to schedule the feature request for the next Scrum sprint.

In the second week, the feature gets pulled into the sprint and the Software Architect and the Software Developer are therefore discussing development details of the feature. In this week, the Product Owner, the Scrum Master and the Consultant are not interested in the discussion of the feature. During the following weeks some issues arise regarding the realized feature that the Software Developer tries to solve. The other TIGER team members are not interested in the feature talk of the stream during this time.

Now, it is assumed that the TIGER team wants to develop a mobile application for the chocolate web application. In the first few days - see Figure 2.8 - the Software Architect of the team does some research on which technology to use and writes the results into the E2SA (week 1). After that initial research, the software architects of the other team in the enterprise catch up and share their ideas, e.g. the advantages and disadvantages of a native mobile implementation (week 2). After this discussion and the subsequent selection of a technology solution, the Software Architect and the Software Developer of the TIGER team implement the solution (week 3). They continue discussing it and are interested in further messages about the technology.

2.8 Recommendation Use Cases

There are two use cases relevant to this work for recommending messages in an E2SA in order to fulfill the scenarios described.

2.8.1 Filtering for Important Messages

The first use case is to filter the message stream for relevant messages only. The user will only see the messages identified as relevant. It is possible to have different levels of relevance, i.e. the user can filter for most relevant, medium relevant or least relevant messages.

2.8.2 Recommending Top Messages per Day or Week

In the second use case the user explicitly chooses to get the most relevant messages by day or week. The user will have a view of the top 10 (or more) messages per day or week, sorted by the relevance score. This makes it possible to get a quick overview of the relevant messages. Depending on the user he may want to check it separately for each day or week.

As a variant of this use case, a user does not want or need messages in his top relevant message list, which he already interacted with. This means,
Figure 2.8: Scenario: Changing interest in a new technology discussion.

For example, if the user replied to a message or got mentioned the message does not need to be included in the top relevant message list anymore. If the user replies to a message he will have read the original message. If the user is mentioned he typically gets a notification (e.g. by email) and the message is already included in a separate view.
2.9 Requirements

Based on the scenario and use cases, requirements can be identified which are relevant for a recommendation algorithm. The requirements are described in the following.

2.9.1 Filtering of Messages Per User

It should be possible to filter the message stream for relevant messages. When filtered, only relevant messages are shown. Therefore, a recommender algorithm is needed that computes a relevance score for a user for a message.

For a common understanding of what relevance in the context of this thesis is, a Relevance Manifest was developed and is presented in Appendix B.

2.9.2 Recommendation of Top Messages per Day and Week

It should be possible to have a list of the most relevant messages per day or week. The recommendation should not consider messages for recommendation in which it is known that the user interacted with the message. There are two options to exclude the messages with interaction: message-based or discussion-based exclusion:

Message-Based Exclusion  Exclude all messages from recommendation with user interaction. This applies to messages the user is author of or is mentioned in.

Discussion-Based Exclusion  Exclude all discussions from recommendation with which the user interacted with. This includes all messages of message-based exclusion. Also, if the user wrote a message in the discussion of the message or got mentioned in a message of the discussion, it will be excluded.

2.9.3 Without explicit User Feedback

An E2SA must maintain a high level of usability to provide a global user acceptance. Therefore, it is not acceptable to let the user explicitly rate a high number of messages or give other feedback that will affect the user acceptance negatively. Hence, the recommendation algorithm must be able to work without explicit ratings and obtain information only from implicit ratings as in [Skr12]. However, implicit ratings are only available after some amount of user interaction.

Therefore, a recommender for computing a relevance score for a message for the user must be able to learn without using explicit user feedback. Instead, the algorithm should learn based on the past experience of the user with the system.
2.9 Requirements

2.9.4 Applicable on Enterprise 2.0 Streams

The recommendation algorithm must be able to deal with an E2S as described in Section 2.6. The algorithm must respect the topic and team structure of an E2S.

2.9.5 Be Adaptive

The algorithm should be able to adapt to new and changing interests. This means there should be no hard-coded user model that slowly or never adapts.

2.9.6 Incremental Integration

If new messages come up, or a new observation is made about the user, the algorithm should be able to integrate the new information incrementally. There should be no overall recomputation of statistics, user model or temporary data for a single new message. If necessary, a re-computation should be done only periodically, e.g. every day or week.

2.9.7 Recommendation in near Real-Time

The recommendation must be in near real-time. This means a computed relevance score or the decision about whether a message is relevant or not must be made within seconds, or up to a few minutes in rare cases. The computation costs for the relevance score must be scalable with the number of users and there should be no full re-computation.

2.9.8 Quality of the Recommendation Algorithm

The goal is to improve the quality of the recommendation algorithm, i.e. to correctly identify the relevant messages for a user. The more relevant messages are found, the better the recommendation algorithm performs, and the higher the quality is.

2.9.9 Non-Requirements

In this section requirements are given that will not be considered in this work.

Implicit Learning from Front-End

There are different options to obtain observations about the user. In Skr12, an approach has been formulated that monitors the front-end interaction of the user and learns a user model from it. In this work, the focus lies mainly...
Problem Analysis

on implicit feedback that can be obtained by direct interaction from the system. Direct interaction is, for example a like of a message, a participation in a discussion or a mention.

Exploration and Diversity

Besides finding interesting messages, it is feasible to find unexpected or serendipitous results as described in [Iaq+08; GDJ10]. Another challenge and ongoing research topic is to increase the diversity of recommended messages. The following simple example should be considered: a user is highly interested in the mobile development of ‘iPhone’ applications. When a new ‘iOS’ is released, this will result in news messages from different sources. All those messages will be scored high for the user, who is actually only interested in the first one, because the other ones are already old news. It will be difficult for other interesting topics to compete against news distributed multiple times through different sources.
Chapter 3

Research Approach

In this chapter the research approach for this thesis is formulated. First, in Section 3.1 challenges for an E2SR are discussed. Then in Section 3.2 the actual approach is formulated before the precise research questions are defined in Section 3.3. Finally, in Section 3.4 issues are given that will not be part of the research approach.

3.1 Research Challenges

There are three main challenges for an E2SR that closely follow the requirements in Section 2.9.

Enterprise 2.0 Social Media Stream Recommender without using Explicit User Feedback

As mentioned before, an E2SR must be able to operate without using explicit user feedback. The challenge is to find a way to use implicit user feedback by fulfilling all other defined requirements, specifically regarding the E2S characteristics.

Adapt to new terms

In Appendix A an analysis on an E2S was conducted. In Section A.3 the change of terms over a three year life cycle revealed that new terms constantly occur. In Section A.4 a simulation of term-based user models presumes that the interest of users is changing heavily within a month. Therefore, an E2SR must be able to adjust to such changes to improve the quality of the recommendation. The E2SR must find a way to react quickly on new and unknown terms and integrate them accordingly.
Short-Term and Long Term Interests

The analysis in Appendix A not only showed that there are new terms and interests but also terms and interests that disappear over time. For example, during a user’s lifetime there will be interests that will not be valid weeks, months or years later but only for a short period of time. This closely relates to the scenario defined in Section 2.7. Not considering this change of interest in a recommender system will probably lead to a decrease in the quality of recommendations.

3.2 Approach

The first goal is to use a recommender that does not need explicit user feedback. The idea is to use the following information about a user to build a user model:

- User is author of a message.
- User is mentioned within the message.
- Message is part of a discussion where the user participated.
- Message is part of a discussion where the user was mentioned.

Based on this information a term-based user model can be learned by extracting terms of the message and then used to match new incoming messages against the extracted terms.

3.2.1 Topic-Specific Enterprise 2.0 Social Media Stream

The main difference of an E2S is the organization of messages in topics. As motivated in Chapter 2 the assumption is that interests of a single user for one topic are different from other topics. This idea has not been covered so far in the current state of the art research.

In Figure 3.1 it is schematically shown how terms are distributed for two topics. There will be terms that both topics share marked as Overlapping Terms. The approach is to learn a user model independently for each topic. If the user interacted with a message the associated terms are learned into the user model. New messages within the same topic and matching terms will be recommended to the user. If a message in another topic occurs but with the same terms it will not be recommended to the user. It is proposed that a topic-specific user model performs better than a global one.

3.2.2 Adapt to new Terms

As mentioned in the research challenges an E2SR should adapt to new terms. In the adaptive scenario new terms occur frequently and the user model
3.2 Approach

Overlapping Terms

Figure 3.1: Overlapping terms in topics (projects).

will not contain all terms to match a new message. The idea is to adapt the unknown terms from somewhere else into the user model. There are two options for User Model Adaptation (UMA): One option is to use user models of similar users. The other option is to use a term of the topic-specific user model of the same user but of a different topic. If a term for adaptation can be determined then the term is added to the user model as long as no other observation about the term is made. This way new unknown terms can be adapted into a user model.

Example

There are two users, user 1 and user 2, each sharing interests in the terms Mobile, Android and iPhone. Now, user 1 reads about the iPad and this term is inserted into his user model. As more messages relating to iPad occur it cannot be determined if those messages are interesting for user 2. The UMA checks the user model of user 1 and adds the entry of the term iPad to the user model of user 2. Then, user 2 will receive a higher score on the messages containing the term iPad.

3.2.3 Short-Term and Long-Term Interests

The UMA adds new unknown terms to a user model. On the opposite side there are interests of users that exist for only a short period of time. The idea arises to identify those short-term interests and only consider them for recommendation for a shorter period of time. Short-term interests are interests that will be forgotten after some time.

As a first step to identifying short-term interests, the change of terms over time must be considered. A short-term interest for a user in a term exists if:

1. There are messages over a certain time span with the term.
3 Research Approach

2. The user is interested only in the term at a specific subset of the time span.

The idea is to maintain a short-term interest user model that keeps the interest for a certain time span. If the interest in the term exists steadily over several time spans, the term will be transferred to a long-term user model. When computing the relevance score for a message, both user models are considered. The long-term user model will be permanent and the short-term model will forget older terms.

3.3 Research Questions and Theses

Based on the approaches described, the following research questions can be formulated that this thesis will research:

- How can the quality of recommendation (measured by F-Scores and TB-MAP) be improved in an Enterprise 2.0 Social Media Stream Recommender (E2SR) without using explicit user feedback?
- How can the quality of recommendation (measured by F-Scores and TB-MAP) be improved by adapting user models to new interests in an E2SR?
- How can a separation of short- and long-term interests improve the quality of recommendation (measured by F-Scores and TB-MAP) in an E2SR?

Following the research approach and questions three theses are claimed:

**Thesis 1**

A content-based recommender with a topic-specific user model will lead to better recommendation quality than one with a global user model.

**Thesis 2**

Adapting unknown terms in the user model by similar users or similar topics will lead to an increase in the quality of recommendation.

**Thesis 3**

Separating short- and long-term interest will lead to an increase in the recommendation quality.
3.4 Out of Scope

There are many challenges for a robust recommendation engine built for enterprise social media streams. The focus of this thesis is the adaptation of user models. Therefore, the following issues will not be part of the research in this work:

- Information extraction is the task of extracting terms or other concepts of a given message. In this thesis, term-based extraction will mainly be used.
- The representation of user models is often related to the available extracted information. In this thesis, term-based user models will mainly be used.
- Related to an adaptive recommendation is the cold start problem, i.e. how to build a user model for new users in a short amount of time and provide good results from the onset.
- Ontologies can be used to define a model about the terms to identify synonyms or inclusions. Defining an ontology can be complex and challenging and depends on the domain of the messages. To be useful at all, ontology must be learned automatically. This is its own research area, and it will not be discussed or used in this work.
Chapter 4

State of the Art

In this chapter, the current research areas that are relevant for this thesis are described. First in Section 4.1, recommender types are described, then in the next Section 4.2, a typical architecture for a recommender system is discussed. In Section 4.3 and 4.4, the main focus is particular information extraction and user models for recommenders. In the Section 4.5, the research areas of concept drift and concept shift are covered. The Section 4.6 describes existing systems that are relevant to this work as well as the combination of algorithms used. Finally in Section 4.7, a summary of the current limitations of the state of the art is presented.

4.1 Recommender Types

In this section the usual classification of recommenders is given. Then, the most important recommender types for this thesis are shortly explained. At the end of this section all recommender types are compared against the requirements.

4.1.1 Classification

The recommender systems area of research has a long history. It is related to the area of Information Retrieval [MRS08]. Information retrieval concentrates mainly on returning the correct results for a given search query. In contrast, recommender systems select and score items predictively for a user.

In [Bur07] and [RRS11], a categorization of recommender systems is given:

Content-based Content-based recommenders use the past ratings of the user to predict new items. They exploit the content of the items to typically learn user models.
Collaborative-based In collaborative-based recommenders items are recommended based on the ratings of similar users. Similar users are users that rated past items similar to a specific user.

Demographic In this type of recommender, demographic information such as the age or citizenship of a user is used to score items.

Knowledge-based In knowledge-based recommenders, domain-specific knowledge is used to understand the needs and interests of the user in order to recommend new items. Case-based and constraint-based recommenders are subgroups of knowledge-based recommenders. In case-based recommenders a similarity measure is applied to retrieve and recommend an item.

Community-based In community-based recommenders the friends of the user and the preferences of those friends are used to recommend new items. In contrast to collaborative recommenders, not all users are used but the social network between the users is emphasised for recommendation. It is interesting to know that [MA04] states that such social network recommendations do not necessarily lead to greater accuracy except for in special situations such as the cold problem. Those types of recommenders are also sometimes called Trust-based Social Recommenders [Zho+12].

Hybrid In a hybrid recommender two or more of the above recommendation types are used.

Other classifications only distinguish between content, collaboration and hybrid based recommenders as in [AT05].

The items suggested by a recommender system are versatile. There are recommender systems that recommend videos, images, music or news. [MLDLR03] provides a list of recommender systems and their application areas. In [Par+12] a classification of recommender systems for application areas and data mining techniques based on research articles has been done. 164 papers have been analysed. Most papers relate to shopping and movie recommenders, consisting of 28% and 21%, respectively. 12% are related to document recommenders, 9% to book recommenders and 6% to music, image and tv program recommenders. Unfortunately [Par+12] does not state which research areas are used in the other 46% of application areas.

For choosing the correct recommender [SG11] defines several evaluation attributes:

- User Preference
- Accuracy
- Coverage
- Confidence
4.1 Recommender Types

- Trust
- Scalability
- Serendipity
- Novelty
- Risk, e.g. stock exchange
- Robustness, e.g. injecting false messages
- Utility (the value the system or user gains from a recommendation)
- Diversity
- Privacy
- Adaptivity

In the context of this work, the main focus is adaptivity. Additionally, the overall recommender system for social media streams must be accurate, scalable, robust, have a high user acceptance and maintain privacy based on topics.

In [TK04] the process of recommender systems is separated into three stages: First, in the data collection phase new items and information about the user are collected and stored. In the second profiling phase a user model is built upon this data. In the third matching phase, new items are recommended.

[MLDLR03] distinguishes the following components of recommenders: the type of user profile in use, the method initially used to generate a profile (also known as the cold start problem), the method of learning a profile, the method of retrieving relevance feedback, and the technique used to adapt the profile.

4.1.2 Content-Based Recommender

[LGS11] divides the recommendation process of content-based recommenders into three steps handled by a separate component:

**Content Analyzer** The analyser takes the content and tries to extract information out of it, transforming the content into a representation that can be used by the other two components.

**Profile Learner** In the learning step available data about a user is collected and then used to construct a user profile.

**Filtering Component** The filtering component selects and ranks relevant content based on the constructed user’s profile by matching it against the content. For determining a rank a relevance score can be computed. As higher the score as higher the rank of the message compared. Alternatively, a Learning to Rank (LTR) [Li11] mechanism is used where typically two messages are compared pairwise to determine the rank of a message.
The advantages of content-based versus collaborative-based recommendation are user independence, transparency and the ability to react to the new item problem \([LGS11]\). In contrast, the disadvantages of content-based recommendation are stated as limited content analysis, over-specialization and reaction to the new user problem.

In \([Li+11a]\), content-based and news-specific recommenders are subdivided into two categories: term-weighting and concept-weighting Recommenders. The first one mainly uses the terms of the news content and matches it against a previously learned user profile. In contrast, the concept of weighting uses ontologies to discover similarity between terms for use in recommendations.

### 4.1.3 Collaborative-Based Recommender

The collaboration-based recommender can be subdivided into three categories: memory-based, model-based and hybrid. Hybrid collaborative algorithms are a combination of both memory and model-based algorithms. The technique used is these algorithms is also named collaborative filtering.

The memory-based technique is a simple and straightforward collaboration algorithm. First, a user similarity matrix is calculated by weighting the rated items across two users. The more items two users rate in the same manner, the higher the user similarity is. Typically the cosine or Pearson correlation similarity is used for computing the user similarity. Next, the user similarity matrix is used to weight new items for a user and the items with the highest score are recommended. The problem of the memory-based algorithm is that it does not perform feasibly in a sparse ratings scenario and it does not perform well in a large dataset.

A subset of memory-based collaborative algorithms are Slope One algorithms \([LM07]\). Slope One algorithms first compute the average differences of the user ratings between an item and the other items. Second, the predicted rating for a user for an item is computed on the average difference. For further references, the collaboration prediction for a user \(u\) and an item \(i\) for Slope One can be simply expressed as:

\[
\text{collab}(u, i) = \text{slopeOnePrediction}(u, i)
\]  

(4.1)

In contrast to memory-based, the model-based collaborative recommenders build models based on the available data. These models consists of latent features that are learned through several data mining or machine learning algorithms. In the review paper \([SK09]\) the following methods for building such models are mentioned: Bayesian Networks, Clustering Models, Latent Semantic models, Probabilistic Latent Semantic Analysis, Multiple Multiplicative Factor, Latent Dirichlet Allocation and Markov Decision Process. Model-based algorithms perform better on sparse or large datasets but - in
contrast to memory-based algorithms - the models are harder to understand and lead to a reduction or loss of useful information.

The main problem for a collaboration-based recommender is the handling of new users and new items - the cold start problem. A lot of algorithms exist to try to solve this problem by stereotyping users or by using content information to make predictions for new items.

### 4.1.4 Community-based Recommender

As mentioned earlier, community recommenders or social recommenders are similar to collaborative recommenders. Typically, with social recommenders the concept of trust is used rather than deploying a plain user similarity [MA04]. The trust is not computed on ratings but on the user’s social network. The findings of [MA04] so far are that the ratings of directly trusted users achieve the smallest error in their setup with an acceptable recall; they perform better in cold start scenarios as a pure collaborative filtering recommender.

### 4.1.5 Comparing Recommender Types against Requirements

In Table 4.1 the general types of recommenders are compared to the requirements defined in Section 2.9. Since collaborative and community recommenders are pretty similar they are treated as one recommender in that comparison. The comparison does not compare a specific implementation of a recommender, instead it compares the concepts of the recommender types against the requirements.

Since all of these are recommenders they fulfill the first two requirements of filtering messages for the user and identifying the top messages.

To work without explicit user feedback from the user model, the collaborative model (e.g. the relationship between user and items) or the knowledge base must be able to build from implicit observations. In a demographic recommender, more or less static demographic information about a user can be supplied by a user profile which is typically available within a system.

All but the demographic recommender can be used in general on enterprise streams. The static information in a demographic recommender will not be flexible enough to get an adaptive recommendation in a stream.

For adaptive and iterative integration the models (user model, collaborative model, knowledge base) of the recommenders must be able to adapt and change incrementally. This is not possible in a demographic recommender.

Similarly, for a near real-time recommendation, the models must be easily query-able. Given a new message, the models must not be too complex or big to provide a quick relevance score. Of course the scoring algorithm itself should not be complex either.
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Content-Based Recommender</th>
<th>Collaborative-/Community-Based Recommender</th>
<th>Demographic Recommender</th>
<th>Knowledge-Based Recommender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without explicit user feedback</td>
<td>Learning a user model must be done from implicit available information.</td>
<td>Most work with explicit feedback (e.g. movie ratings). Collaboration network must be learned or built from implicit available information.</td>
<td>Using profile information</td>
<td>Necessary to create and maintain a Knowledge Base which is challenging without explicit user or expert feedback.</td>
</tr>
<tr>
<td>Applicable on E2S</td>
<td>Are applicable.</td>
<td>Are applicable.</td>
<td>Demographic Information will not be enough to recommend in depth.</td>
<td>Yes</td>
</tr>
<tr>
<td>Be Adaptive</td>
<td>The user model must be able to reflect the change of interest.</td>
<td>The collaboration model must be able to reflect the change in time.</td>
<td>Barely, since the demographic information used for prediction is fixed.</td>
<td>Knowledge Base must be adaptive.</td>
</tr>
<tr>
<td>Incremental Integration</td>
<td>User model must be incrementally maintainable.</td>
<td>Collaborative model must be incrementally maintainable.</td>
<td>Yes, since only demographic information is used.</td>
<td>Knowledge Base must be incrementally maintainable.</td>
</tr>
<tr>
<td>Recommendation in near Real-Time</td>
<td>Matching a message to a user model must be executed in near real-time.</td>
<td>Computing the prediction for a message using the collaboration model must be done in near real-time.</td>
<td>Matching a new message against the demographic information must be done in near real-time.</td>
<td>Matching a new message against the knowledge base must be done in near real-time.</td>
</tr>
</tbody>
</table>

Table 4.1: Comparing recommender types against requirements.
4.2 Typical Recommender Architecture

In Figure 4.1 the architecture of a typical recommender system is shown. There are recommenders that will use different components as shown in the figure but most recommenders use the components in some form or another.

The relevant components are divided into two sides: on the right side is the application-specific component and on the left side the recommender component. There are two types of communication between the application-specific component and the recommender component:

**Observation** The recommender component is notified about events in the application that will have an impact on the recommendation. Typically events are interactions of the user with the system, e.g. rating a message or object.

**Scored Items** The recommender components provide the actual recommendation, typically a set of scored items where the score reflects the strength of the recommendation.

The recommender component consists of several sub components:

**Information Extraction** This component analyses the content to extract information from an item. This can be terms and key phrases as well as features.

**Data Layer** This component manages the data access to other items, users or user models. Typically, the data layer will also include caching strategies to avoid a bottleneck.

**Data Mining** The data mining component mainly supports the learning and scoring components. It will compute relationships and clusters that are necessary for the other components.

**Recommendation Model Learner** This component learns user models by observations and will also adapt user models to improve the scores.

**Scorer** This component will compute a relevance score for an item per user based on the user model and extracted information from the other components.

When processing an item some kind of information extraction first takes places. Features will be extracted from the item and stored with the item. For example, in a content-based recommender terms or n-grams are determined by the content of an item.

Based on the existing items and their extracted features or descriptions a data mining component will find new information. The mined results
are typically not user-specific and stored. Often the data mining stage is computationally expensive and will not be executed for every new item but periodically, e.g. each day or week.

Examples for data mining in the recommender component are:

1. Determining the term frequency based on all extracted items.
2. Building a network between user, items and extracted features.
3. Clustering users, messages or items.

With the mined results, the items and their descriptions a recommender model is build or learned. This model can be user-specific. Typically, in a content-based recommender a term-based user model is learned based on the ratings or observations of the user. In a collaborative recommender one central model reflecting the user- to-item relations or connections is created.

Finally, the scorer component will take the recommender model and match new items to compute a relevance score. Typically, the item description is used and matched against the recommender model of the user. The relevance score is then provided to the application-specific component which will use it for retrieval.
4.3 Information Extraction for Recommender

For content-based recommenders the step prior to recommendation is to extract information from the items. In the area relevant to this work, the items are short messages containing text that may or may not contain Hypertext Markup Language (HTML) tags.

One goal of information extraction is to extract terms, n-grams or char grams. The typical approach is to first clean-up the text by removing unwanted characters or HTML tags. In order to extract terms the text is split into words and then stop words are removed. Then stemming is often applied; this is an algorithm that reduces a term to the word stem.

Another variation is to extract n-grams instead of stemmed terms. N-grams is a sequence of \( n \) terms in the text. Similarly, n-char grams are extracted in \( n \) subsequent characters.

An extension of this term-based information extraction is the usage of ontologies. Terms can be matched within the ontology which can then provide more semantic information about the term and hence the message.

Other approaches in information extraction use the principle of tagging. The main difference between tags and terms is that tags do not necessarily need to be included in the text. Typically, a message dataset with assigned tags is trained so that the resulting classifier is used to tag the new incoming message. The problem with this approach is that a clean and large amount of training data must be available. This is a drawback for changing interests because the tagging classifier does not easily adapt to changing content, as there is no constant relearning option.

Named Entity Recognition (NER) [Chi+99] is an information extraction approach in which the entities are extracted from underlying documents. Examples of entities are author, location or product name. The instance of such entities is the actual value mentioned in the document such as \textit{author=Joe Smith, location=Los Angeles, product name=iPad}. Entity recognition algorithms tend to work better with large documents than with small and short messages.

There are also information extraction algorithms that use mainly statistical models such as Probabilistic Latent Semantic Indexing (PLSI) [Hof99] or Latent Dirichlet Allocation (LDA) [BNJ03]. In the latter it is assumed that a model exists with hidden features and each document (or message) consists of a probabilistic distribution of certain features. Those features are then learned in an unsupervised manner.

4.4 User Models

User models are the most common form of recommender model and maintain some kind of user specific features. Recommendations are computed
based on the user model by determining a relevance score. Typically, this computation is a matching of the feature or description of the item and the user model.

### 4.4.1 Representation of User Models

The representation and structure of a user model depends heavily on the available data that has already been extracted. In the literature, the following representations are used:

- **N-Grams or Terms** This user model consists of several n-grams or terms or even a combination. A score for each entry represents the confidence or interest of the n-gram or term to the user. This can also be the vector space model.

- **History-based** In a history-based user model the concrete examples of user interaction are stored. For example, in a web application a history-based user model will contain all URLs accessed by the user.

- **Neural Network** The user model itself can be a neural network that is constructed in such a way that it can classify incoming items. The process of scoring items is then reduced to apply the neural network to the features of the item for recommendation.

- **Rule-based** In this user model several rules are maintained that are then used on new items during the scoring stage.

There are many ways to represent a user model and combinations of several representations are often used. Choosing the right user model depends on the type of items to recommend, the learning algorithms available and other recommendation constraints.

### Vector Space Model

The vector space model is a representation of a text (such as a document or message) or as simple as a search query. In most cases, the vector space model consists of terms for which each term a value is calculated. The value represents the degree of membership of that term to the text. Besides terms, n-grams or char grams can also be used in a vector space model. For each term statistics such as the term frequency and the inverse term frequency can be computed. Typically, the inverse term frequency or the Term Frequency - Inverse Document Frequency \( \left( \frac{tf}{idf} \right) \) is used as term weights to give a higher relevance to rare terms.

The advantage of the vector space model is that a similarity between two vectors can easily be computed. This is commonly known as the cosine similarity. With this measurement a similarity can be computed between
4.4 User Models

search queries, documents and messages. The cosine similarity for two vectors $A$ and $B$ can be defined as:

$$\text{cosine}(A, B) = \frac{\sum_i A_i \cdot B_i}{\sqrt{\sum_i (A_i)^2} \cdot \sqrt{\sum_i (B_i)^2}}$$  (4.2)

4.4.2 Profile Generation

There are several ways to generate a user model. The simplest and easiest way is to use an empty user profile that can be incrementally updated. However, this way will lead to a low recall in the starting phase of the system. Another way is to do a manual profile generation in which the user provides a source with his interests. In social recommenders a source often used to generate a new profile is an existing account, e.g. Delicious [AV014].

Another way of generating a profile is to use stereotyping. In this technique, clusters of users are computed to form stereotypes. The cluster description - the stereotype - is then used as a profile for a new user.

4.4.3 Learning of User Models

The learning method of a user model depends on the user model representation and the available data. Input for a learning algorithm includes:

- Observations about the user that can result in implicit ratings or even explicit ratings given by the user
- History information, which is the user’s past interactions
- Profile information that uses age, location and other demographic features of the user

One learning method is a reinforcement learner: It takes feedback from a recommendation; if it is positive, it adapts positively to the causal data and if the feedback is negative it adapts negatively. For example, if a message with the term ‘iPad’ is recommended and the user rates it positively the value of that term will be slightly increased. Another way of learning a user model is using classifiers such as a Bayesian Classifier.

4.4.4 Adaptation of User Models

In the literature User Model Adaptation (UMA) is used with different definitions. In the first definition, UMA is performed on the user model, meaning the user model is changed. That definition will be used within this thesis.

Other literature uses a user model for adapting an existing system. In this case, adaptation refers to the personalization of a system and a user model
helps with that. That means the adaptation itself is more the process of scoring than of changing a user model. This definition applies to the research area of Adaptive Hypermedia [Bru01] which concentrates on personalization and adoption of portals (like museum web portals or shopping sites). This is not the definition used in this thesis.

[VBW04] describes a general UMA engine. It defines push, pull and hybrid UMA: push performs adaptation when storing the user model, pull performs adaptation when information is required about the user and hybrid does both.

[MLDLR03] distinguishes four types of UMA. The first is manual user model adaptation. An expert or the user himself has the potential to access and change a user model. Example [CC03] concentrates on authoring systems that help users understand and adapt a user model. The second type of adaptation is to add new information by a learner or an adaptation engine. The third type is gradual forgetting, which is a way of dealing with changing interest. Concept drift and gradual forgetting is discussed in detail in the next section 4.5. The fourth adaptation is natural selection: there, genetic algorithms are used to select the user model with a high level of fitness.

4.5 Concept Drift and Shift

As stated in the problem formulation, a recommender must adjust to changing interests. This is called concept drift (for gradual changes) and concept shift (for abrupt changes).

In general there are two approaches which are independent from the actual learning algorithm. One approach uses the principle of a time window as in [WK92]. Within a window not all of the examples of a learning algorithm are considered, only those which match the window. The window moves over time in such a way that the learning algorithm will work with different examples and hence adapt to a concept change. However, the main issues here are choosing the correct window size or using an adaptive window size as well as controlling the computational complexity. As stated in [CGM03] the problem of too short of a window size is adaptive but not stable behavior. In contrast, the problem of long window sizes is stable but not adaptive behavior. To take control of the computational expenses for a window of time the learning algorithm must be able to learn incrementally: that is, react according to examples that are added to the current window as well as examples that are removed from the current window without a full recomputation [CGM03] [WK92].

The second approach is based on gradual forgetting. Here, the examples will lose their impact over time: older examples will have lower impact than more recent ones. Mathematically, this is implemented by using an exponential time decay factor.
A basic algorithm that deals with concept drift using window size is Flora2 [WK92]. Here three sets are maintained within a window: one with consistent descriptions, one with complete but inconsistent descriptions, and one with negative descriptions. Upon a change of the window and underlying example the examples are incrementally integrated into the sets.

To overcome the limitations of a fixed window size some algorithms use an adaptive window size. In [KR98] concept drift is detected by constantly monitoring the accuracy, recall and precision. Those measurements are then compared to an error from a moving average and if necessary the window size is adapted. The problem with this approach is that training examples are necessary to compute the errors and more parameters need to be tuned.

An enhancement for this approach is used in [CGM03] which uses the concept of Statistical Quality Control [Mon97] to detect concept changes and adapt the window size. The main idea is to use user feedback to compute an error and detect if the error is in control or if concept drift occurs. If a drift occurs and is detected the window size is narrowed. [DR09] uses a more sophisticated statistic test to overcome the limitations of fixed measurement. Here, the statistical tests adapt the underlying data to detect concept drift better than fixed error measurement.

Another interesting approach is used in [TLY09]. Here concept drift is detected by the standard methods described above and decision trees are built for a particular window size. However, the goal of this approach is to present the end users with an understanding of an occurring concept drift. Therefore, the decision trees between two windows are compared and rules are extracted that reflect the concept change. An example of such a rule is:

\[
\text{If } (\text{fever} = "\text{no} \Rightarrow \text{yes}") \text{ and } (\text{Work} = "\text{Chicago} \Rightarrow \text{Shanghai}") \text{ and } (\text{cough} = "\text{no} \Rightarrow \text{yes}") \text{ then } (\text{diagnosis} = "\text{healthy} \Rightarrow \text{SARS}").
\]

In [CHL13] a Gibbs sampling algorithm is used to detect changes in time in a collaboration-based movie recommender.

4.6 Existing Relevant Systems

In this section existing recommender systems are described. As stated in section 4.1.1 there are many different recommenders with different scenarios but this section concentrates only on systems related to social media streams recommendation. These include news and social collaboration recommenders.
4.6.1 IBM Connections

[Guy+10] and [GRR11] describe a recommendation engine that is used in IBM Lotus Connections (now IBM Connections [IBM14]), a social software for enterprises. Following the classification in Section 4.1.1 this recommender is a community-based recommender that maintains an interaction graph. Additionally, a gradual forgetting function is applied in the scoring process but not in the user profile.

In the profiling stage an interaction graph based on the social interaction is constructed. This graph consists of nodes representing tags, persons and items. Examples of such items are wiki pages and blog entries within the system. Next, each node is linked to a value representing the degree of the relationship and the interaction between two nodes. If a user visits a wiki page or a blog entry the link value increases. If two users are in the same project or are marked as co-workers the link value increases. Based on that graph the user profile is constructed. For each user, the top-$n$ similar users and tags are determined and stored in the profile. When recommending items the user profile is evaluated against an item and new items are favored using an exponential decay factor in the scoring process. The main influence for the recommendation is the extensive use of the user structure.

In [GRR11] they extend this model by using the data of an activity stream. They take the items of the stream where the user is actor and integrate it into the interaction graph. Also, places are extracted from the items and used for recommendation. They do not recommend items in the stream but use the information to recommend wiki pages, blogs, etc.

Applicability on E2S: The structure of users that is used and need for the recommendation is not given or observable in a common E2S. In an E2S connections between users can be observed based on interaction in the same discussion, the same topic or users of the same group. Users who are editing or viewing the same wiki page cannot be transferred to an E2S. A message is only written by one user. Whether other users view or read this message cannot be directly observed by an E2S. Hence, an interaction for several users for the same message is not possible. Besides their user structure their approach depends on the tags of items. Often in an E2S tags are used more loosely and are seldom administrated for a single message. They also use only the top user profile elements, which makes the approach not useful for filtering of messages.

4.6.2 Google News

A collaboration-based recommender which must fulfill high performance needs for the Google News [Goo14] website is described in [Das+07]. There, three approaches are combined: collaborative filtering using MinHash clustering, PLSI and covisitation counts. MinHash is a probabilistic clustering
4.6 Existing Relevant Systems

method in which two users are assigned to the same cluster if the items both users voted for overlap. Since MinHash itself is not scaleable for the amount of news and user, they used Locality Sensitive Hashing (LSH) \(^{[GIM99; AI08]}\). In PLSI a relationship between items and user is learned, then co-visibility builds a graph based on items a user has visited within a time span.

In order to be able to compute recommendations within a reasonable time the first two approaches are implemented as a MapReduce\(^{[DG08]}\) algorithm. The principle of MapReduce is to split the computation task between several machines with distributed data. The separate results are then combined into a final result.

**Applicability on E2S**: The approach is based on optimizing the recommendation for large sets of messages and users. It finds a few relevant messages from a large message pool. The recommendation is based on the interaction and ratings of the similar users. Hence, a message is only recommended if some interaction took place. This does not happen for most of the messages in an E2S. First, the number of users that will interact with a single message is much lower than in a public news application. Typically, only a handful of users will interact with a message in an E2S which makes it nearly impossible to rely only on collaboration-based interactions.

### 4.6.3 ALOE, OrganiK: Recommendation of Latent Topics

Latent topics in this context are latent features of news articles and not topics as in the context of an E2S. The idea of those latent topics is that each news article can be described by a set of such hidden topics with an associated weight for each topic.

**ALOE**

In \(^{[Sch10]}\) a enterprise social media sharing platform named ALOE was used to infer latent topics and recommend them to users. In the evaluation users rated recommended topics as either interesting or not.

**Applicability on E2S**: The approach recommends latent topics in an E2S. The approach does not recommend messages in an E2S and can therefore not be used as an E2SR.

**OrganiK**

In \(^{[CMA12]}\) an enterprise social software named OrganiK is used to recommend tags and resources based on latent topics. However, it is not a user-specific recommender. It recommends resources or tags based on a given resource.
Applicability on E2S: *Organik* cannot be used directly for filtering on messages since it only recommends based on a specific selected tag or resource.

### 4.6.4 SCENE, PENETRATE: Personalized News Recommendation

In [Li+11b] a personalized news recommendation called SCENE (*SCalable two-stage pErsonalized News rEcommendation*) is described. It uses a two stage recommendation with a hybrid user model. An extension of this system called PENETRATE (*PErsonalized NEws recommendaTion framework using ensemble hierarchiClustering*) is described in [Zhe+13].

In [Li+11b] they state that collaborative filtering alone is not applicable for news recommendation because it is not effective for news items, since no collaborative data is available. Hence, using only collaborative filtering, new items are hard to recommend.

In the first stage of the recommendation process topics are recommended. Topics in this context are latent topics as described before in 4.6.3. The idea of those topics is that each news article can be described by a set of such topics with an associated weight for each topic. For determining topics a clustering algorithm is applied to the news data. This clustering algorithm uses [LSH] to build small groups of news items, then organizes those groups hierarchically into clusters. Each cluster is described by probabilistic language models such as [PLSI] and [LDA]. Those clusters then form topics which are recommended to the user using the cosine similarity. In PENETRATE this clustering is extended by an ensemble hierarchical clustering algorithm.

In the second stage the recommended topics are used and items that increase the quality are selected for the user. Here, quality means the item is similar to the underlying news topic, similar to the user and has low diversity in the selected items. The selection is done by the principle of *Budget Maximum Coverage Problem*, where the budget is the number of items. Finally, the selected items are ranked by time and relevance.

The user model of SCENE and PENETRATE is three-fold and consists of weighted topics, a list of similar users and a list of named entities. This model is maintained based on the reading history of the user.

In the evaluation news articles are skipped that have less than 10 clicks per day. Both systems reach an $F_1$-Score ranging from 0.25 to 0.35.

**Applicability on E2S:** Both systems use the reading history of the users. Whether or not a user read a message in an E2S cannot be determined surely. Again, the collaboration between users is used for recommendation. The interaction in an E2S will not be the same in intensity for a message as in such a news system. If in an E2S messages with less than 10 interactions are removed, nearly all messages will be removed, thereby making a recommendation useless.
4.6.5 Tag-Based Recommender

In social communities, tags have been used widely to annotate images, links, news, movies or documents. Recommenders use these tags for recommendation [NDH06; DD09]. Typically a user tags an item himself and then those used tags are used to find other resources.

A sophisticated approach to recommend documents based on tags using semantics is described in [DD10]. They define a document similarity function which uses a document score, a tag similarity and a user-tag affinity. The document score is computed based on the popularity and representativeness of a tag. The popularity is based on the number of overall occurrences and the tag representativeness is based on the term frequency. The user-tag affinity measures how often a user uses a specific tag. They focus then on tag similarity by detecting tag synonyms and equated tags using the WordNet [Pri14] database and ontologies. The results of this implementation show that these simple semantics lead to a slight improvement in accuracy.

**Applicability on E2S:** Tag-based recommendation only works if resources to recommend are well tagged and the tags are used for several items. If a tag is only used once it will not be helpful in recommendation. Also, the user must constantly tag resources or a method must be found to infer tags for a user. In an [E2S] tags are only rarely used and not in an intensive manner as in the common tag-based recommender approaches [NDH06; DD09]. Also, users in an [E2SA] very rarely tag other messages. Both these issues make tag-based recommendation unfeasible for [E2SR].

4.6.6 Yahoo News Recommender

In [Li+10] the recommendation algorithm of the Yahoo News [Yah14] front page is described. The algorithm itself is more or less a demographic recommender. The user profile is a predefined feature set containing gender, geographic and history information. The history consists of the categories previously visited or used within the platform.

News items are represented by their URL and tags which are maintained by editors of the platform.

The scoring itself is based on a multi-armed bandit which is an optimization algorithm based on probabilities. The algorithm sequentially chooses news articles for a user based on the user’s profile and the news article description.

**Applicability on E2S:** Demographic information is insufficient information to recommend messages in an [E2S]. Also, tags are maintained by an editor. Having a user manage tags of all messages in an [E2SR] is unrealistic and would make an [E2SR] unfeasible for productive use.
4.6.7 Hermes: Semantic-Based Approach

Hermes [Sch+10] uses a semantic-based approach to recommend news items of Really Simple Syndication (RSS) feeds. From the news items, events are detected and will update a graph-style knowledge base. The user selects interesting concepts in the knowledge base, then related concepts are presented to the user. They apply a manual event validation by an expert to enforce the quality of the detected events.

**Applicability on E2S:** As before, having a user or expert to intervene in the recommendation process makes the approach unfeasible for an E2SR.

4.6.8 Twitter Recommenders

Using Twitter data for recommendation has attracted attention in research over the last few years. For example [HDD11] uses the frequency of retweets to determine and predict the popularity (or interestingness) of a tweet. A similar approach using content-based methods is used in [Nav+11] and another approach with the same goal using an incremental Naive Bayes model is applied in [Alh+12].

In [HBS10] a hybrid approach is used: A content-based approach uses the past tweets of a user as input and a collaborative-based approach the followed users. In the best configuration they reach an average precision of nearly 0.3.

In [Dia+12] the Twitter hash-tags are used to find the top relevant messages for a user out of a large portion of Twitter data. They focus on the real-time recommendation for the large data but only use the hash-tags as a recommendation basis. Relying solely on hash-tags will only help to find a few top relevant items, and will also miss all those without hash-tags or with rarely-used hash-tags.

**Applicability on E2S:** Computing the popularity or the chance of a retweet for a tweet is not reflecting the relevance of the tweet for the user. Besides that, retweeting or reposting a message is only rarely used in an E2S and will not have enough potential for useful recommendations. Recommending on the follow structure is also not a promising approach for an E2SR. The interests of users within an E2S differs from topic to topic and also the users involved in the topic are different from topic to topic. This is unrepresentable by the follow structure a user can build within Twitter.

4.6.9 Facebook

There is no scientific publication available that describes the Facebook recommendation algorithm. Many sites such as [Kin10] propose that Facebook uses or used EdgeRank. In EdgeRank news items are objects and for each
interaction (such as commenting, tagging) an edge is created between the objects.

To compute the score \( \text{score}_o \) for an object \( o \) in the news stream, all edges of the object are used. Per edge, three components are evaluated. First, \( u_e \) as the affinity score between edge creator and the user who views the stream is used. Second, a weight \( w_e \) based on the type of edge (comment, tag, like, etc.) is used. Third, a time decay factor \( d_e \) is used which is lowered as an edge gets older, based on the creation time of the edge. With these components the score can be computed as follows:

\[
\text{score}_o = \sum_{\text{edge}} u_e w_e d_e
\] (4.3)

However, it seems that EdgeRank in this form is no longer used within Facebook. In [McG13] the author Matt McGee interviews Lars Backstrom, Engineering Manager for News Feed Ranking at Facebook and McGee states the following:

Lars Backstrom, Engineering Manager for News Feed Ranking at Facebook, estimated that there are as many as "100,000 individual weights in the model that produces News Feed." The three original EdgeRank elements - Affinity, Weight and Time Decay - are still factors in News Feed ranking, but "other things are equally important," he says.

Later, McGee summarizes:

"EdgeRank is a thing of the past, and it’s been replaced by a machine learning-based algorithm that, as Backstrom says, "only ever gets more complicated."

So this means Facebook is using something more complicated than EdgeRank with far more than 100,000 weight factors.

In [Bou+13] a LTR [Li11] approach is used to learn a ranking model and use it to rank messages in a stream. They applied their algorithm in a Facebook application but it is not the official Facebook implementation. There, they use message, source and user features for their algorithm and use training data obtained by user ratings to train the ranking model. They reached a Precision@1 of 0.273 which means that 27.3% of the first retrieved elements per user in the stream are relevant. They also reached a Precision@5 of 0.190 which means that 19% of the top five elements are retrieved as relevant.

**Applicability on E2S**: Since there is no clear description of the algorithm it is uncertain how it could be used for an E2S. The main issue, as with Twitter, is the absence of topics and the different interest within different topics. This probably makes the Facebook recommendation algorithm not directly applicable for an E2S. Additionally, the LTR approach of [Bou+13] used training data which is not directly available in an E2S.
4.6.10 Comparison of Existing Systems

In Table 4.2 the described existing systems are compared against the different technologies and algorithms used:

**Recommender Type** In this column the type of recommenders as described in Section 4.1.1 are specified. Most of them use a combination of content- and collaboration-based approaches.

**Item Type** The item type defines the items that are recommended. These are mostly news messages or web pages except for the latent topic approaches.

**Information Extraction** Some approaches use PLSI or LDA to detect latent topics and use them later in a learning or scoring step. Other systems do not use any information extraction. They typically just use the available data for recommendation or user model learning.

**Data Mining Algorithm** If data mining algorithms are used then it is mostly some form of clustering algorithms. Typically, the result of the information extraction is used to identify clusters and then assign clusters to user or vice versa.

**User Model** Typically, the user model contains a cluster membership (if clusters have been identified before), a latent topic membership (if latent topics have been identified) or extracted terms. In some cases the user models are the connections in a user-to-message structure (e.g. for memory-based collaborative-based recommenders).

**Learning Algorithm** All except the Yahoo approach [Li+10] learn incrementally. If clustering is used the clustering is mostly computed offline in periodic cycles.

**Scoring Algorithm** The scoring algorithm is in most cases some similarity between the message representation and the user model, e.g. term similarity.

**Concept Drift and Shift** Only a few systems actually use a specific form to handle concept drift. Of course, if incremental learning is used the system adapts over time if new observations are made. However, only two systems take additional actions to handle concept drift.

**Evaluation Method** Most of the systems are evaluated against Precision, Recall and $F_1$-Score.

**Applicable on** As already described separately for each system none of them are applicable to an E2SR. The reason is summarized in this column.
4.7 Limitations of State of the Art Approaches for Enterprise 2.0 Social Media Stream Recommender

A lot of research has been done on collaborative methods. However, collaborative recommendation using users-to-messages relations is not feasible. For new messages no collaborative information is available after the message first appears in the system.

The first two requirements of filtering and recommending top messages are fulfilled by nearly all recommenders except those which recommend latent topics. As described before, some of the recommenders build their models based on implicit available information, while others need explicit feedback to recommend messages. In the latter case those algorithms become unfeasible for an E2SR.

Only a few of the recommenders are used and evaluated within an enterprise, but none of them on E2S. In [GRR11] activity streams are only used to enhance the recommendation of related items, not of the messages in the stream itself. As shown, all existing systems lack the ability to be applied to an E2SA (see Section 4.6). There is no algorithm or existing system that fulfills all the necessary requirements. They lack the ability to deal with the characteristics of an E2S or the existing approaches are too specific to a certain task. Also, none of the approaches deals explicitly with changing interests in a social media stream or an E2S.

Hence, the approach described in Chapter 3 is novel because this approach specifically deals with the characteristics of an E2S including the fulfillment of all defined requirements. The proposed approach will use existing algorithms, e.g. content-based algorithms, and will combine and adapt them in such a way as to make them applicable for an E2SR. Furthermore, part of the proposed approach considers the change of interests over time.
<table>
<thead>
<tr>
<th>System</th>
<th>Recommender Type</th>
<th>Item Type</th>
<th>Information Extraction</th>
<th>Information Extraction</th>
<th>User Model</th>
<th>Learning Algorithm</th>
<th>Scoring Algorithm</th>
<th>Concept Drift and Shift</th>
<th>Evaluation Method</th>
<th>Applicable on E2S</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Connections</td>
<td>Community-Based</td>
<td>Wiki Pages</td>
<td>Social Graph</td>
<td>Similar Users, Tags, and Items</td>
<td>Incremental</td>
<td>Aggregation</td>
<td>Decay Factor</td>
<td>True Positive</td>
<td>No: Not Stream-Based.</td>
<td></td>
</tr>
<tr>
<td>Google News</td>
<td>Content-, Collaborative-Based</td>
<td>News</td>
<td>PLSI</td>
<td>MinHash Clustering, Covisitation, LSH</td>
<td>Collaborative Model, not user-specific</td>
<td>Expectation Maximization</td>
<td>Cluster Association to Users, Covisitation Score</td>
<td>No: Not Stream-Based, focuses on User Structure.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latent Topics</td>
<td>Content-Based</td>
<td>Latent Topics in Social Media</td>
<td>Stemming, ( f - idf )</td>
<td>Clustering Weighted Terms per Topics</td>
<td>Incremental</td>
<td>Similarity</td>
<td>( F_1 )</td>
<td>No: Recommends Latent Topics not Messages.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Personalized News Recommendation</td>
<td>Content-, Collaborative-Based</td>
<td>News</td>
<td>PLSI, LDA, k-Shingles</td>
<td>Clustering Similar Users, Weighted Topics, NE</td>
<td>Incremental</td>
<td>Budget Maximum Coverage</td>
<td>Windowing</td>
<td>Limited: Only applicable if enough Interaction occurs on a Message. Not the case in an E2S.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tag-Based</td>
<td>Community-Based</td>
<td>Web pages</td>
<td>Clustering, Tag Graph and Semantics Tag to Cluster Relation</td>
<td>Incremental Cluster Matching</td>
<td>-</td>
<td>-</td>
<td>No: Not Stream-Based, only applicable if Items are well tagged.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>Approach</td>
<td>News Source</td>
<td>Manual, Categories</td>
<td>Clustering</td>
<td>Features and Cluster Relations</td>
<td>Reinforcement</td>
<td>Multi-armed Bandit</td>
<td>Relative</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------</td>
<td>------------------------------</td>
<td>--------------------</td>
<td>------------------</td>
<td>-------------------------------</td>
<td>---------------</td>
<td>-------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Yahoo</td>
<td>Demonographic-Based</td>
<td>News</td>
<td>Manual, Categories</td>
<td>Clustering</td>
<td>Features and Cluster Relations</td>
<td>Reinforcement</td>
<td>Multi-armed Bandit</td>
<td>Relative</td>
<td>No: Not Stream-Based, only uses static Profile Information for Learning.</td>
<td></td>
</tr>
<tr>
<td>Twitter</td>
<td>Content-, Collaborative-Based</td>
<td>Tweets</td>
<td>Stemming,</td>
<td>Similar Users, Weighted Terms</td>
<td>Incremental Term Similarity</td>
<td>-</td>
<td>-</td>
<td>Relative</td>
<td>No: Only applicable on Twitter Structure.</td>
<td></td>
</tr>
<tr>
<td>Facebook</td>
<td>Collaborative-Based</td>
<td>Messages</td>
<td>Social Interaction Graph</td>
<td>Connections between Users and Messages</td>
<td>Incremental Aggregation by weighted Factors</td>
<td>-</td>
<td>-</td>
<td>Relative</td>
<td>Limited: Stream-Based but not Focus on Enterprises.</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of the state of the art existing recommender systems.

- a Probabilistic Latent Semantic Indexing (PLSI)
- b Locality Sensitive Hashing (LSH)
- c Precision (P)
- d Recall (R)
- e Term Frequency - Inverse Document Frequency ($tf-idf$)
- f $F_1$-Score ($F_1$)
- g Latent Dirichlet Allocation (LDA)
- h Named Entity Recognition (NER)
- i Named Entities (NE)
- j Natural Language Processing (NLP)

4.7 Limitations of State of the Art Approaches for ESR
Chapter 5

Recommender Concept

In this chapter the recommender concept to solve the research question of Chapter 3 is developed.

First, in Section 5.1 the basic architecture for an E2SR is described. Then, in Section 5.2 the idea of learning from features is explained. The recommendation algorithm and process is described in Section 5.3. The approach that uses a content-based recommender is described in detail in Section 5.4. The advanced content-based approaches of user model adaptation and short-term interest are described in Section 5.5 and 5.6 respectively.

The collaborative-based approach is described in Section 5.7. In Section 5.8 a genetic algorithm is introduced for parameter optimization of the algorithms.

At the end in Section 5.9, the requirements of the problem analysis (see Section 2.9) are used and compared against the recommender concept of this chapter.

5.1 Basic Architecture

In Section 4.2 a typical recommender architecture was described. The architecture for the described E2SR will also be based on this architecture. The recommendation component and the interfaces as used throughout this work are shown in Figure 5.1.

The interfaces of the recommendation component as well as the subcomponents are described next. At the end of this section architectural requirements and how they can be achieved are described.

5.1.1 Interfaces

The recommendation itself should be as independent as possible from a specific application logic. Therefore, two interfaces are defined for use in communication between the recommender and the application component.
5 Recommender Concept

Figure 5.1: Overview of recommendation architecture.

Observation Interface

For a social media system such events are:

1. User A likes message X.
2. User A follows User B.
3. User A unfollows User C.
4. A new message has been created.
5. A new topic has been created.
6. A message has been deleted.
7. User A received access to topic C.

An event belongs to one or both of the following categories:

---

1 This list is not complete and depends on the actual social media system in which to integrate.
5.1 Basic Architecture

Learning Events (Observations)  An event of this category is used to learn about a specific user or a group of users. For example, the event ‘User A likes message X.’ will lead to an update of the user model of user A with the terms of the message.

Scoring Events  An event with this category influences the scoring process. For example the event ‘A new message has been created.’ will start the scoring process.

Scoring Interface

The observation interface is mainly for the communication of data from the application to the recommender component. The way back is defined by the scoring interface. In this interface the computed recommendation scores are sent to the application. In this thesis, the recommendation score is a relevance score whereby one relevance score is computed for one user for one message.

Message Retrieval Interface

The computed relevance scores must be made available somehow. The message retrieval interface will provide the relevance scores for messages, a list including the most relevant messages, or whether a message is relevant or not.

5.1.2 Dependent Components

There are three components providing basic functionality that the recommendation components depend on.

1. Data Layer

The data layer provides the data model for messages, users, scores, etc. It persists or gets the necessary data. The implementation itself can use a database, a file system, or distributed storages.

2. Communicator

The communicator component allows asynchronous communication between the components, or within a single component. It also queues communication messages as long as the receiver is busy. This takes care of communication and avoids overload. A typical implementation is a message queue that stores unprocessed messages in memory or on disk as long as a message is not processed.

A communication message is used to exchange information for processing between components, e.g. to start the recommendation process or to
handle an observation. A communication message can contain a content message. When the recommendation process should be invoked a communication message is created that contains a content message. The content message is the original message in an E2S. The created communication message is then sent via the Communicator component to start the recommendation process.

3. Information Extraction

The information extraction component takes a message and extracts terms or n-grams from this message, removes stop words and cleans the text. It then returns a set of terms describing the message.

5.1.3 Recommendation Subcomponents

The interaction between the subcomponents are described in detail in the next sections. Here, only a short overview of the components themselves (see Figure 5.1) is given.

- The Retrieval component provides the functionality of the message retrieval interface. It uses the data layer to filter for messages for a user based on the computed message score.

- The Scorer component takes a new message and computes a relevance score for each user. The learner component uses observations of a user to learn and maintain a user model.

- The User Model Adaptation component is invoked by the scorer to adapt a user model on new terms. It uses a User Similarity Computer to find similar users which are then used by the user model adapter to perform the adaptation.

- The Short-Term component maintains a short-term user model by the User Model Maintenance and also a Long-Term Detector that detects terms that occur in specific patterns in the short-term model and stores them in a long-term model.

5.1.4 Architectural Requirements

The main requirements for the architecture are performance (for fast recommendation) and the integrability to allow the recommendation to be adapted to different environments.

Performance

In order to support high performance, the recommendation must be able to scale horizontally so that the components can be distributed in multiple
instances across different server nodes. This can be achieved by loosely coupling the components and subcomponents of the recommendation. This also means that the communication between the components must be asynchronous so not to block another component from execution. This can be achieved by using a message queue implementation.

One bottleneck in social media application is the access to the data storage as evaluated in [Mar12]. If the application is based on a central data storage the application will not be able to scale horizontally. To overcome this limitation the data storage must be accessed using an extensive caching layer.

Integrability

To reach the performance goals and the integration described, several architectural patterns are relevant. One pattern already mentioned is the Event [Gam+95] pattern. This pattern helps to loosely couple the components. The Event Listener will only need to register and handle the event. The Event Sender - in this case the application-specific component - only needs to send the event.

Another pattern that will be applied is the Adapter [Gam+95] pattern. This pattern allows a separation of concerns providing a general interface with several implementations for different purposes. For example, the data layer component will provide a general interface to the recommendation component. The recommendation component itself will not be responsible for caching or maintaining the data but the data layer component will be. The recommendation component will also be independent of the actual implementation in the data layer. This way different data providers can easily be used.

For processing a stream-based architecture the Command [Gam+95] pattern is useful. Here, several Commands are defined that will execute a specific coherent logic forming a processing chain. Those commands are then executed in a chain to give the desired results. The advantages of this chain are the centralization of logic in the commands and the separation of incoherent logic into different commands.

5.2 Basic Algorithm

For each message, features can be computed implicitly without using user feedback. Some features are user dependent, while others are user independent. User independent features only need to be computed once per message. User dependent features need to be computed for each user. It is obvious that the features can be used for scoring, but how this will be done must be determined. Besides using the feature for scoring the
5 Recommender Concept

The idea is to use the features for learning and maintaining the recommender. This principle is shown in Figure 5.2. Here, the scoring process extracts features for an incoming message per user shown as Message User Features. The Recommender Model depends on the recommender used (see Section 5.2.2). Using the extracted features and the recommender model the scoring process will compute a relevance score for each user for a message. Also, using the extracted features, the learning process will maintain the recommender or user model.

5.2.1 Features

In Table 5.1 the features which, given a message, can be easily obtained are given. Everything except the Discussion Root Feature are user-specific features. Those features are boolean features with a score of either 0 or 1. The features can be used to compute a relevance score directly and also to determine if a message should be used to learn a user model.

Example

An example for message and computed features is shown in Table 5.2. In this table two discussions are shown. For each message the Mention feature and Discussion features are listed including the users for whom the feature exist.

5.2.2 Recommender Model

If a specific recommender (e.g. content-based or collaborative-based) is used, a recommender model must be maintained. For a content-based recommender this will be a user model. In contrast, for the collaborative
### 5.2 Basic Algorithm

Feature Description Per User

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Per User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discussion Root Feature</td>
<td>The message is not a reply to another message.</td>
<td>x</td>
</tr>
<tr>
<td>Author Feature</td>
<td>The user is the author of the message.</td>
<td>x</td>
</tr>
<tr>
<td>Mention Feature</td>
<td>The user is mentioned in the message.</td>
<td>x</td>
</tr>
<tr>
<td>Like Feature</td>
<td>The user liked the message.</td>
<td>x</td>
</tr>
<tr>
<td>Discussion Participation Feature</td>
<td>The user participated in the discussion the message is part of.</td>
<td>x</td>
</tr>
<tr>
<td>Discussion Mention Feature</td>
<td>The user is mentioned in a message that is part of the discussion of the message</td>
<td>x</td>
</tr>
<tr>
<td>No Discussion Participation Feature</td>
<td>The Discussion Participation Feature is not present.</td>
<td>x</td>
</tr>
<tr>
<td>No Discussion Mention Feature</td>
<td>The Discussion Mention Feature is not present.</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 5.1: Message features in the scoring process.

57
<table>
<thead>
<tr>
<th>Parent Message No.</th>
<th>Parent Message No.</th>
<th>Author</th>
<th>Message Content</th>
<th>Mentions</th>
<th>Discussion Mention</th>
<th>Discussion Participation</th>
<th>Discussion Root</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>Lisa</td>
<td>@karl We need team Tiger to implement the external user group synchronization feature soon!</td>
<td>Karl</td>
<td>-</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Karl</td>
<td>no problem @lisa. Will do.</td>
<td>Lisa</td>
<td>Lisa</td>
<td>Lisa</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Lenny</td>
<td>@karl No! Stop. Wait for the next sprint planning for a final decision for user group sync.</td>
<td>Karl</td>
<td>Lisa, Karl</td>
<td>Lisa</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>Karl</td>
<td>@lisa @lenny ok, user group synchronization is scheduled for the next sprint.</td>
<td>Lenny, Lisa</td>
<td>-</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Emily</td>
<td>@lisa finished implementation.</td>
<td>Lisa</td>
<td>Lenny, Lisa</td>
<td>Karl</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 5.2: Example for features in the scoring process for two discussions.
The recommendation process consists of two main processes, the Scoring Process and Learning Process, which are described in detail. Both processes and the User Model Adaptation Process is shown in Figure 5.3.

5.3.1 Scoring Process

The main trigger for the scoring process is a new message. The output of this process is a set of relevance scores, i.e. a score for each user who has access to the topic. The relevance score is a number in $[0..1]$ where 1 expresses the highest relevance and 0 the lowest relevance.

In some cases the UMA process will be invoked and may change the user model. It is likely that the scoring process will lead to more accurate results once the UMA process runs for the same message. On the other hand, it is possible that the UMA process will not be able to run in near real-time. If the scoring process waits for the UMA process to finish, the computed score would not be available in near real-time; however, this is crucial for the recommendation engine. The other way around would lead to inaccurate results because the scoring would not be executed on the newest user model and therefore the quality of the overall recommendation would decrease.

The solution for this problem is to execute a two-stage scoring process:

Ad Hoc Scoring In the first scoring stage, the scoring process is started directly after a new message has been observed. The score is computed independently of the learning and UMA processes. The score will be available quickly.

Rescoring The second scoring stage will be started once the user model adaptation process finishes. The rescoring process can also be limited to only consider users with a changed user model. A complete re-scoring for all user is not necessary. The new computed score will take more time but will be more accurate. Rescoring is not necessary if the ad hoc scoring already returned a high relevance score.
Figure 5.3: Detailed recommendation process.
5.3 Recommendation Process

Information Extraction

In the information extraction step the content of the message is broken down into separate terms. First, the content of message is cleaned by non-word characters. Also, if the content is only HTML the content will be extracted. Second, tokens will be extracted by splitting the content on white space characters. Third, stemming is done for each token to get the final terms to be used for further processing.

Compute Message-Specific Feature

In this stage, the message-specific Discussion Root Feature is computed.

Per User Iteration

The next stages are completed for those users who have access to the message.

Per User: Compute User-Specific Features

In this stage the user-specific message features are computed.

Per User: Compute Message Relevance Score for User

To compute a relevance score the question is which feature to use and how to use it for scoring. In the pre-analysis in Appendix A.2.2 a decision tree has been used. There for a node in the decision tree the percentage of relevant and not relevant messages can be determined. The depth of the tree is two at maximum, meaning that two features are used. So the scoring itself can be represented rather simply and does not need a complex algorithm.

Let $F_{m,u}$ be the tuple of computed feature scores for message $m$ and user $u$, then $F : m,u$ can be expressed formally as:

$$F_{m,u} = (f_1, f_2 \ldots f_n)$$ (5.1)

The feature score is in $[0..1]$. For a boolean feature the value is either 0 for false or 1 for true. Let $fw_i$ in $[0..1]$ per feature $i$ be fixed feature weights. The score for each message can be computed by using the maximum of the feature value and weight:

$$score(m,u) = \max \{fw_i \cdot f_i | 0 \leq i \leq n, f_i \in F_{m,u}\}$$ (5.2)

The feature weights can then be determined by an optimization algorithm. The outcome will be more or less an ordering of the features: the higher a feature weight gets, the higher the resulting relevance score will be. With the negative features No Discussion Mention Feature and No Discussion Participation Feature Equation (5.2) will be able to model the behavior found in the pre-analysis.
Per User: Invoke User Model Adaptation

After computing the relevance score, the process can be invoked asynchronously. The details are described in Section 5.5.

Per User: Invoke Learner

In this stage the learning process is triggered if necessary. This is described in the next.

5.3.2 Learning Process

The learning process is responsible for observing information about a user and maintaining the recommendation model for this user. To start the process an observation must be made about a user. This observation can be explicit, e.g. a like of a message by a user or an explicit rating by the user. Also, it can be an interaction with a message, e.g. the user wrote a message. In both cases the information can be used for learning the recommendation model.

Determine Interest and Terms

The goal of this step is to compute or define an observation score \( \text{obs}(m, u) \) for a user \( u \) and a message \( m \) that can be used to update the recommendation model. In cases where the observation is explicit, a fixed value can be used for the observation \( \text{obs}(m, u) \).

In order to obtain an observation score from implicit interaction out of the scoring process, a decision must be made whether a message and the associated terms should be used for learning, and to which extent the feature should be used learned.

First, only messages will be used for learning if the compute score \( \text{score}(m, u) \) fulfills a Score-To-Learn threshold. Second, learning feature weights \( \text{lfw}_i \) will be used (similar to Equation 5.2) to determine the observation score that will be used for learning:

\[
\text{obs}(m, u) = \max \left\{ \{\text{lfw}_i \cdot f_i \mid 0 \leq i \leq n, f_i \in F_{m,u}\} \right\} \quad (5.3)
\]

Third, the terms of the message the observation score has been obtained from are used in the next stage: the update of the recommendation model.

There is also the option to learn from parent or all parent messages. So, if the message \( m \) is an answer to a parent message \( pm \) than the two options are possible:

Learn from Parent Message The terms of the parent message \( pm \) are used as well for updating the recommendation model.
Learn from All Parent Messages The terms of the parent message $pm$, and of the parent message of the parent message and so on are used as well for updating the recommendation model.

Update Recommendation Model

The message or the terms of the message and the obtained observation score are integrated into the recommendation model. In the content-based approach this means updating the user model, and in the collaboration-based approach updating the user to message connections.

5.4 Content-Based Approach

This section describes a content-based approach that will maintain a user model in the learning process, and match the message against the user model in the scoring process.

General User Model

First the user model is introduced. The user model maintains all the terms of the users; the second user model is a short version of the full one containing all the interesting terms.

Definition 1. The User Model $UM$ contains a set of terms $T$ and each term $t$ has a term score $s$ assigned:

$$UM_u = \{(t, s)|t \in T, s \in [0, 1]\}$$

(5.4)

The term score $s$ represents the users' interest in the term. To refer to a single user model entry or user model term score for term $t$ of user $u$ in the user model $UM$ the syntax $um_u(t)$ is used.

Time-Binned User Model

Let $\tau = [start, end]$ be a time interval with $start$ and $end$ being the start and end date of the interval, respectively. A time-binned user model can be defined as:

Definition 2. The Time-Binned User Model $UM_{\tau,u}$ maintains term weights based only on observations within a time interval $\tau$.

For the recommendation process it is relevant to define the length of the interval that will be used.
Confidence

In the scoring process the user model $UM$ is used to match the terms of the message to the terms of the user model. If the user model is new it will not match or only partially match the terms of the message. A distinction must be made if the computed score is either a lack of interest or whether there is not enough information to decide.

The confidence defines the number of terms in the user model matching the message to the overall number of terms. Let the function $\text{term}(m)$ return the terms for message $m$ then, the confidence of a computed message score for the message $m$ within a user model. It can be defined as:

$$\text{confidence}(UM_u, m) = \frac{|t \in UM_u \land t \in \text{term}(m)|}{|\text{term}(m)|} \quad (5.5)$$

The confidence is the ratio of terms in the user model matching the terms of the message to the overall number of terms of the message. The confidence states how many terms of the message are defined within the user model. Only those terms tell the difference between an unknown term or a non-interest term.

Topic-specific User Model

As defined in 2.4.3 each message belongs to a topic. The user model defined in Equation 5.4 can be learned independently per topic. This is the same as treating the same term $t$ as a different term if the term occurs in a different topic. Formally, a topic-specific user model can be defined as:

**Definition 3.** The **Topic-Specific User Model** $UM_{TP,u}$ contains a set of terms $T$ for each topic $k \in TP$ and each term $t$ has a weight $w$ assigned:

$$UM_{TP,u} = \{(t_k, w) | t \in T, k \in TP, w \in [0,1]\} \quad (5.6)$$

Within a topic-specific user model the learned term weights are independent between each topic. That is, the interests within one topic are treated as independent from other topics.

5.4.1 Learning User Models

The user model must be learned by the features observed. Each observation is based on a message and a message contains extracted terms. There are two strategies to learn a user model that will be used in this thesis.

Term Count Learning Strategy

In this strategy the strengths of the observations per term is summed up and set in relation to the overall number of observations. The strength
determines how relevant (positive) or irrelevant (negative) the message of the observation is for the observed user.

With \( \text{obs}(u) \) as the set of all observation messages of user \( u \) and with \( \text{obs}(u, m) \) as the observation strength for the user \( u \) for the message \( m \), the user model term score \( um_u(t) \) can then be computed as:

\[
um_u(t) = \frac{\sum_{m \in \text{obs}(u) \cap \text{t} \in m} \text{obs}(u, m)}{\sum_{m \in \text{obs}(u) \cap \text{t} \in m} 1}
\] (5.7)

This learning strategy never forgets, and the order of observation does not matter; it will always lead to the same user model term score \( um_u(t) \).

### Incremental Learning Strategy

In this strategy the user model term score for a user is either incremented or decremented based on the observation strength.

Let \( \beta \in [0..1] \) be a threshold for a neutral observation. For each term \( t \) of a message \( m \) of an observation the following is computed. The first equation is used if \( \text{obs}(u, m) \geq \beta \) and the second if \( \text{obs}(u, m) \leq \beta \):

\[
\begin{align*}
\text{um}_u'(t) &= \text{um}_u(t) + \alpha \cdot (1 - \text{um}_u(t)) \cdot \text{obs}(u, m) \quad \text{(5.8)} \\
\text{um}_u'(t) &= \text{um}_u(t) - \alpha \cdot \text{um}_u(t) \cdot (1 - \text{obs}(u, m)) \quad \text{(5.9)}
\end{align*}
\]

Here \( \alpha \) is a learning factor with \( \alpha \in [0..1] \). The higher \( \alpha \) is, the faster the user model term score will change to the last values. The order of integrating the observations is important in this learning strategy. Also, it does forget old values.

Another variation of this strategy is to use the current user model term score as threshold to determine if the entry value should be incremented or decremented. That is the first equation \( \text{(5.8)} \) is executed if \( \text{obs}(u, m) \geq \text{um}_u(t) \) and the second \( \text{(5.9)} \) otherwise.

### 5.4.2 Term Weights

There are different options for handling term weights. A term weight represents how unique and strong a term is. The more often a certain term is used, the less important the term becomes for prediction. In document retrieval the [inverse document frequency][MRS08] is often used. With \( d \) as a document consisting of a set of terms, and with \( D \) as the set of all documents, the inverse document frequency can be used:

\[
\text{idf}(t, D) = \log \left( \frac{|D|}{|\{m|m \in D \land t \in d\}|} \right)
\] (5.10)

Sometimes, the Term Frequency - Inverse Document Frequency \( \text{tf-idf} \) is also used to determine the importance of a term (or word) within a document.
5 Recommender Concept

In an E2SA the messages are mostly short and terms are not repeated often within one message, so the inverse document frequency is sufficient.

It is quite simple to get the Inverse Message Frequency (imf) for a term by mapping the inverse document frequency to messages:

\[
\text{imf}(t) = \log \frac{|M|}{|\{m | m \in M \land t \in m\}|} \tag{5.11}
\]

If a topic-specific user model is being used, the imf will only be computed on the message within the topic.

5.4.3 Content Match Feature

If the user model has been learned and is steadily maintained, it can be used to match against new incoming messages.

Again, for each message terms can be extracted. For those terms the user model entries are then used to compute a content match score. The score states how similar the user model is to the message, and how much interest the user has in the message.

The user model term scores 4.4.1). For two such vectors the similarity can be computed by the cosine similarity (see Equation 4.2). This cosine similarity can be used for the CMF. Here the message vector contains the term weights (imf) for the terms that exist in this message). This term message vector is matched against the user model vector. This leads to the definition of CMF:

\[
\text{CMF}_{\text{Basic}}(u, m) = \frac{\sum_{t \in m} u_m(t) \cdot \text{imf}(t)}{\sqrt{\sum u_m(t)^2 \sqrt{\sum \text{imf}(t)^2}}} \tag{5.12}
\]

One question is how to deal with unknown terms in the user model. There are two options. The first option is to treat the values as zero, that is \(u_m(t) = 0\) if \(t\) does not exist in the user model. The second option is that those terms are ignored when computing the CMF. Both options are used in the evaluation for optimization.

Example

An example for computation of the CMF is shown and explained in Figure 5.4. The example uses the messages introduced in Table 5.2.

5.4.4 Cold Start

For the content match feature to work it needs to learn from previous messages to build up the user model and the term weights. If the recommendation algorithm is integrated into an existing application, it can just use old messages to learn from them, as long as the features can be inferred.
5.4 Content-Based Approach

1. Example user model for Emily before message 4:

<table>
<thead>
<tr>
<th>Term $t$</th>
<th>Term Score $um_{Emily}(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule</td>
<td>0.40</td>
</tr>
<tr>
<td>next</td>
<td>0.40</td>
</tr>
<tr>
<td>sprint</td>
<td>0.15</td>
</tr>
</tbody>
</table>

2. Extracted terms of message 4 with example Inverse Message Frequency ($imf$):

<table>
<thead>
<tr>
<th>Term $t$</th>
<th>$imf(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>0.05</td>
</tr>
<tr>
<td>group</td>
<td>0.10</td>
</tr>
<tr>
<td>synchronization</td>
<td>0.80</td>
</tr>
<tr>
<td>scheduled</td>
<td>0.22</td>
</tr>
<tr>
<td>next</td>
<td>0.03</td>
</tr>
<tr>
<td>sprint</td>
<td>0.30</td>
</tr>
</tbody>
</table>

3. Compute Content Match Feature:

$$CMF_{Basic}(Emily, m_4) = \frac{\sum_{t \in m_4} um_{Emily}(t) \cdot imf(t)}{\sqrt{\sum um_{Emily}(t)^2} \sqrt{\sum imf(t)^2}}$$

$$CMF_{Basic}(Emily, m_4) = \frac{0.4 \cdot 0.05 + 0.4 \cdot 0.10 + 0.15 \cdot 0.80}{\sqrt{0.4^2 + 0.4^2 + 0.15^2} \sqrt{0.05^2 + 0.10^2 + 0.80^2}}$$

$$CMF_{Basic}(Emily, m_4) = 0.278$$

Figure 5.4: Example for a user model and for a computation of the $CMF$. In the first step it is assumed that the user model of Emily only contains three terms with the given term score. Second, the terms for message 4 of the example in Table 5.2 are extracted. Finally in the third step the terms of the message and term scores of the user model are matched to compute the $CMF$. 
If no messages exist, the user model can be learned from new messages and will need some time to build up a user model to make useful recommendations.

5.5 User Model Adaptation Approach

In an E2S, unknown terms arise very often in a constant rate (see Appendix A.3). Collaborative mechanisms will not work directly by applying them to new messages (see Section 4.1.2 and 4.1.3) and typically a scoring algorithm will not adapt quickly to the new terms. Therefore, an approach will be established which will extend the content-based approach and adapt user models for unknown terms based on similar user models or topics.

The idea of the User Model Adaptation (UMA) is to adapt a user model using terms of other user models or topics then there is a low confidence on compute relevance score. For the adaptation, three issues have to be tackled:

1. When to adapt a user model?
2. How to choose user model(s) or topic(s) to adapt from?
3. How to integrate a term into the adapted user model?

The process of the User Model (UM) is shown in Figure 5.3 (Section 5.3) and each step is described next.

Invoke User Model Adaptation

The UMA will adapt the user model based on a selected message to change the computed relevance score. The adaptation is triggered by a new message during the scoring process if the following criteria are fulfilled:

1. The confidence (see Equation 5.5) of the message for the user model is below a confidence threshold $\zeta$.
2. The relevance score of the message is below a fixed score threshold $\rho$.

Mathematically, that can be summarized as:

$$\text{crit}_{UMA} = \text{confidence}(UM_u, m) < \zeta \land \text{score}(UM_u, m) < \rho$$  (5.13)

The confidence criterion will only consider messages that do not have enough confidence and a UMA is highly promising for having an impact on a new computed relevance score. If the relevance score of the ad hoc scoring stage has already identified the message as interesting, then an adaptation will not change the message score significantly and is not necessary.

The input for the UMA process is then:

- The user $u$ for whom the adaptation is taking place.
- A set of terms that exist in the message $m$ of the scoring process but not in the users’ user model: $t \in m \land t \notin um_u(t)$. 

68
Identify Similar User or Topics

There are three variants of UMA that are described in Section 5.5.1, 5.5.2, and 5.5.3. The first variant takes user models from similar users and the second variant takes terms of the user model from the same user but from different topics. The third variant uses a combination of similar users and topics.

Based on the similar users or topics only the top most similar are chosen, or all similar users or topics are used.

Determine Terms To Adapt From

For the selected users and topics of the step before, it will be determined whether or not an unknown terms exists. If not, the UMA stops, since there is nothing to adapt from.

Compute Adapted Term Score

For each of the unknown terms, the term scores of the found terms from the different similar users or topics are merged. The merged term score is the adapted term score.

Adapt User Model

The adapted term score is stored in the original user model.

Rescore Message With User Model

The scoring process is repeated for the user that the user model was adapted for.

Example

An example for UMA is shown and explained in Figure 5.5. This example is based on the example presented in Figure 5.4.

5.5.1 User Model Adaptation by Similar Users

User models of similar users can be consulted to adapt a user model for an unknown term.

The main issue here is to use a suitable user similarity. The idea is to use the existing terms between two user models to compute a user-model-based similarity.
1. Compute the confidence for message 4 for the user model of Emily:

\[
\text{confidence}(UM_{Emily}, m_4) = \frac{|t \in UM_{Emily} \land t \in \text{term}(m_4)|}{|\text{term}(m_4)|}
\]

\[
\text{confidence}(UM_{Emily}, m_4) = \frac{3}{6} = 0.5
\]

2. Assume a confidence threshold of \( \zeta = 0.75 \) and a score threshold of \( \rho = 0.5 \). Then, \( \text{crit}_{UMA} \) (see Equation 5.13) is fulfilled and the UMA is invoked for Emily’s user model with three unknown terms: user, group and synchronization.

3. Assume that all three unknown terms exist in Lisa’s user model with a score of 1.0. All three will be used for adaptation of Emily’s user model.

4. Adapted user model of Emily:

<table>
<thead>
<tr>
<th>Term</th>
<th>Term Score ( um_{Emily}(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule</td>
<td>0.40</td>
</tr>
<tr>
<td>next</td>
<td>0.40</td>
</tr>
<tr>
<td>sprint</td>
<td>0.15</td>
</tr>
<tr>
<td>user</td>
<td>1.00</td>
</tr>
<tr>
<td>group</td>
<td>1.00</td>
</tr>
<tr>
<td>synchronization</td>
<td>1.00</td>
</tr>
</tbody>
</table>

5. Adapted CMF:

\[
CMF'_{\text{Basic}}(Emily, m_4) = \sqrt{\sum_{t\in m_4} um_{Emily}^t(t) \cdot imf(t)}
\]

\[
CMF'_{\text{Basic}}(Emily, m_4) = \sqrt{1.05^2 + 0.10^2 + 0.80^2 + 0.40^2 + 0.03^2 + 0.15^2 + 0.30^2}
\]

\[
CMF'_{\text{Basic}}(Emily, m_4) = 0.673
\]

Figure 5.5: Example for applying UMA. This example is based on the example shown in Figure 5.4. To invoke the UMA the confidence is computed (1). If the confidence is less than a defined threshold the UMA is invoked with the unknown terms (2). Now, it is assumed that the user model of Emily is similar to the one of Lisa (3) and the terms of Lisa’s user model are adapted to Emily’s user model (4). With the adapted user model the adapted \( \text{CMF} \) is computed (5). The value of the \( \text{CMF} \) for the message is raised from 0.278 (without adaptation) to 0.673 (with adaptation).
User-Model-Based Similarity

One option is to compute the cosine similarity between two user models. It is likely that the user models will only match a small percentage and this will lead to a similarity near zero if both only contain a small set of identical terms.

A better idea is to compute the similarity only on interest terms; i.e., the terms where the user model term score $um_u(t)$ succeeds an interest term $i$ threshold:

$$\text{interestterms}(um_u) = \{t | t \in um_u \land um_u(t) > i \} \quad (5.14)$$

The user-model-based similarity can then be computed using the Jaccard similarity [TSK05]. The Jaccard similarity for two sets $A$ and $B$ is defined as follows:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (5.15)$$

Here, the number of elements that occur in both sets is compared to the number of all unique elements in both sets.

Using the Jaccard similarity on the interest terms set, the user-model-based similarity for two users $u$ and $v$ can be computed as follows:

$$\text{usersim}(u, v) = \frac{|\text{interestterms}(um_u) \cap \text{interestterms}(um_v)|}{|\text{interestterms}(um_u) \cup \text{interestterms}(um_v)|} \quad (5.16)$$

A high similarity is reached, if the user models share a high number of identical interest terms.

Compute Adapted Score

Using the user similarity the adapted user model score can be computed using the weighted average for all users that contain the term $t$:

$$um'_u(t) = \frac{\sum_{v \in U \land v \neq u \land t \in um_v} um_v(t) \cdot \text{usersim}(u, v)}{\sum_{v \in U \land v \neq u \land t \in um_v} \text{usersim}(u, v)} \quad (5.17)$$

Using Top-$n$ Similar Users

When using all users to adapt the score, only the user models that contain the term $i$ as an interest term will be considered at all. If there is only one user, when the term score will be taken no matter how similar the user actually is.

To avoid this behavior the idea is to use only the top-$n$ most similar users for adaptation. Only the user models of the top-$n$ users are consulted to find the adapted term score. If none of the top-$n$ users contain the term as interest
no adaptation will take place. This is a more restrictive strategy than using all users to compute the adapted term score.

Let \( \text{topusersim}_n(U, u) \) be the \( n \)-highest user similarity for user \( u \) (excluding user \( u \) itself), then the top-\( n \) similar users can be expressed as follows:

\[
\text{topuser}_n(U, u) := \{ v | v \in U \land v \neq u \land \text{usersim}(v, u) \geq \text{topusersim}_n(U, u) \}
\] (5.18)

The adapted score using the top-\( n \) users is only computed with the top-\( n \) users that have the term to adapt in the user model:

\[
\text{um}'(t) = \frac{\sum_{v \in \text{topuser}_n(U, u) \land t \in \text{um}_v} \text{um}_v(t) \cdot \text{usersim}(u, v)}{\sum_{v \in \text{topuser}_n(U, u) \land t \in \text{um}_v} \text{usersim}(u, v)}
\] (5.19)

### 5.5.2 User Model Adaptation by Similar Topics

If a topic-specific user model is used then terms can be adapted from other topics. The idea is the same as before but instead of finding similar users, topics are picked which contain the term to adapt. In this adaptation variant only terms within one user model are used. No similar users or similar user models are used to find a term for adaptation. Only the user model that will be adapted is used.

Again, the Jaccard similarity can be used to compute the topic similarity. The terms of the message of the topics are used for similarity computation. The computation of the topic similarity is limited to only past messages, e.g. from the last month. This way the topic similarity adapts automatically with time. Let \( \text{terms}(k) \) be the set of all terms that exist in the topic \( k \) since a time span, then the topic similarity for two topics \( k \) and \( l \) can be computed as:

\[
\text{topicsim}(k, l) = \frac{|\text{terms}(k) \cap \text{terms}(l)|}{|\text{terms}(k) \cup \text{terms}(l)|}
\] (5.20)

### Compute Adapted Score

Now, let \( t_k \) and \( t_l \) be the same term \( t \) in the topics \( k \) and \( l \), respectively. With \( TP \) as the set of all topics, the adapted term score based on the topic similarity can be computed as:

\[
\text{um}'(t_k) = \frac{\sum_{t \in TP \land t \neq k \land \text{um}_u(t) > 0} \text{um}_u(t) \cdot \text{topicsim}(k, l)}{\sum_{t \in TP \land t \neq k \land \text{um}_u(t) > 0} \text{topicsim}(k, l)}
\] (5.21)

The main difference from the user similarity approach is, that the terms are adapted from the same user model but different topics.
5.5 User Model Adaptation Approach

**Using Top-n Similar Topics**

Similar to the top-n user similarity, only the top-n topics can be used to determine the adapted term score. Let $toptpsim_n(TP,k)$ be the n-highest topic similarity for topic $k$ (excluding topic $k$ itself), then the top-n similar topics can be expressed as:

$$toptopics_n(TP,k) := \{ l \mid l \in TP \land k \neq l \land topicsim(k,l) \geq toptpsim_n(TP,l) \}$$ (5.22)

The adapted term score using the top-n topics is only computed with the top-n topics which have the term to adapt in the user model:

$$um_u'(t_k) = \frac{\sum_{l \in toptopics_n(TP,k) \land um_u(t_l) > 0} um_u(t_l) \cdot topicsim(k,l)}{\sum_{l \in toptopics_n(TP,k) \land um_u(t_l) > 0} topicsim(k,l)}$$ (5.23)

If none of the top-n topics contain the term, no adaptation will take place.

**5.5.3 User Model Adaptation by Similar Users and Topics**

It is also possible to determine the terms to adapt from both users and topics at the same time. This works only if a topic-specific user model is used.

The problem is that both user similarity and topic similarity cannot be combined equally. The idea is to use a voting principle. Let $\gamma$ define the number of maximum votes. The the top-n similar users and top-n similar topics can vote to adapt the term. As a vote the top-n similar users that contain the term $t$ is used. The number of all top-n user votes can be expressed as:

$$numTopNUsers(u,t) = |v \in topuser_n(U,u) \land um_v(t) > 0|$$ (5.24)

Similarly, the top-n similar topics that contain the term $t$ can vote. The number of all topics top-n topic votes is computed as:

$$numTopNTopics(u,t_k) = |l \in totopic(TP,t) \land t_l \in l|$$ (5.25)

The new adapted score is then the ratio of the votes to the number of maximum votes $\gamma$. It is limited to a maximum term score of 1 in case more votes than maximum votes are provided. The combined similarity is then computed as:

$$um_u'(t_k) = \min \left(1, \frac{numTopNUsers(u,t_k) + numTopNTopics(u,t_k)}{\gamma} \right)$$ (5.26)
5.5.4 Adaptation and Learning

If a user model is adapted, the respective user model term score will be marked as adapted. During the learning process the adapted term score $\text{um}_u(t)$ will be ignored and replaced by the actual term score $\text{um}_u(t)$. Hence, once an observation is made for that term the adapted weight will be dropped and replaced by the normal weight.

5.5.5 Change of Similarity over Time

Both similarities change over time. As the user model evolves the user similarity will be different, as well as the topic similarity. To be more accurate, the adapted score can be recomputed periodically to reflect the latest similarity.

A smart way to achieve this is to check during the ad hoc scoring phase whether the an adapted term score is used. In that case the UMA process can be invoked to recompute the adapted term score.

5.6 Short-Term and Long-Term Approach

In Section 5.4 a static user model is defined. Terms contained in this user model are fixed and do not change unless an observation is made or an adaptation takes place. As introduced in Chapter 2, short-terms interests should be identified and treated differently to achieve a higher recommendation quality.

The assumption is that there are user interests that only exist over a short period of time (from a day within a week following the results of the pre-analysis in Section A.4.1). After that time span the interest fades away and recommendations based on this interest should not be made.

The idea is to use a short-term user model that keeps the interest for a user only for a short period of time. Only if the interest of the term occurs repeatedly will it be recognized and transferred into a long-term user model.

The short-term model can be modeled as time-binned user model as defined in Section 5.4. The messages are separated into time bins, where each time bin has a specific length as shown in Figure 5.6. The time-binned short-term user model contains the terms of a message where the observation was made within that time bin as shown in Figure 5.7. The number of bins is limited, and old bins will be removed from the user model. The short-term user model only consists of the latest time bins.

The time-binned user model only works with the term-count user model learner. Here, the values of the single time bins can be easily combined to get the term score. This is not easily possible for the incremental user model learner.
5.6 Short-Term and Long-Term Approach

5.6.1 Time Decay

If an user interest is only for a short period of time it fades away as time passes. Commonly, a decay factor is applied that will decrease the relevance score the older a message gets. In the scenario of an E2SR the decision and computation of relevance must be done in real-time (see requirement in Section 2.9.7). Also, the presentation of the relevant messages is necessary within the same day or at least the same week. A decay factor based on the message only helps if it is likely that messages that are weeks old will still be relevant. This is typically not the case for E2S or social streams in general.

The idea is to apply a decay factor to the term score in the user model when the CMF score is computed. In general, with $\Delta t$ as time difference and $\lambda$ as time decay factor, this can be expressed as:

$$um_{u,\text{decay}}(t) = e^{-\lambda \cdot \Delta t} um_u(t)$$  \hspace{1cm} (5.27)

The time difference used is the creation time of the message and the last change of the term in the user model. The main parameter is how fast the score decays: For a better interpretation, let $\kappa$ define the time difference when the decayed term score reaches half of the original value. For example, for $\kappa = 1$ day a term score of 1 will have a value of 0.5 one day after the
term was last changed ($\Delta t = 1\text{day}$). With this definition $\lambda$ can be expressed by $\kappa$:

\[
\frac{1}{2} \cdot um_u(t) = e^{-\lambda \cdot \kappa} \cdot um_u(t)
\]

\[
\kappa = -\frac{\ln(\frac{1}{2})}{\lambda}
\]

With $\kappa$ as half value cut off parameter this gives the decay function:

\[
um_{u,\text{decay}}(t) = e^{\ln(\frac{1}{2}) \frac{\Delta t}{\kappa}} \cdot um_u(t) \tag{5.28}
\]

This can be simplified to the final decay function:

\[
um_{u,\text{decay}}(t) = 0.5 \frac{\Delta t}{\kappa} \cdot um_u(t) \tag{5.29}
\]

### 5.6.2 Long-Term User Model

In addition to the short-term model, a long-term model is also maintained. Here, two question arise: First, when does a term gets added to the long-term model? Second, when matching a message to the user models how to combine the two user models?

#### Long-Term Detector

To add a term to the long-term user model it must be available in the short-term user model for a period of time, having multiple occurrences. An occurrence is an interest term (see Equation 5.14) within on time bin. The following options to add a term to the long-term user model are used:

- **Periodic Occurrence** The term occurs in the user model at least $n$ consecutive time bins for $o$ times. In Figure 5.8 a periodic occurrence for 3 periodic occurrences is shown. The minimum distance defines the distance between two consecutive occurrences.

- **Permanent Occurrence** The term occurs in at least $n$ consecutive time bins with a maximum of $g$ gaps. In Figure 5.9 a permanent occurrence is shown. The gaps allow detection of permanent occurrences if the occurrence is missing for some time bins.
5.6 Short-Term and Long-Term Approach

If a long-term interest is detected the user model term score \( um_u(t) \) is transferred from the short-term to the long-term user model. The long-term interest is then maintained by the term count learning strategy (see Section 5.4.1). If a new observation occurs and is integrated by the learning process also the long-term user model is changed. No terms are added by the learning process to the long-term user model but only from the long-term detector which runs periodically.

Merging User Models

To compute a CMF both of the user models must be merged in some way. This can either be done by merging the term scores and then compute the CMF, or by computing a CMF for each user model and then merge the scores.

In both cases, an aggregation strategy is used which is one of the following:

1. Use the maximum value of both user models.

2. Use a weighted average. The weight \( \omega_{UM} \in [0..1] \) defines to which extent the short-term user model is used.
Merging of CMF

When merging the user models, the CMF can be computed independently per user model and then aggregated by using the maximum value or a weighted average. With $M_u$ as the set of user models for user $u$ and $\omega_{um}$ as a fixed weight, the weighted average then is:

$$CMF_{\text{Weighted}}(u, m) = \frac{\sum_{um \in M_u} \omega_{um} \cdot CMF_{um}}{\sum \omega_{um}}$$ (5.30)

The weight $\omega_{um}$ is defined per user model type, and is independent from the users, the weight defines the impact in the weighted average. Accordingly, instead of using the weighted average the maximum value can be used for merging:

$$CMF_{\text{Max}}(u, m) = \max(\{CMF_{um}(u, m) | um \in M_u\})$$ (5.31)

Merging of User Model Term Scores

In the term-based merging each term of the user model is taken separately and merged with the term of the other user model, using the aggregations strategy. The merged terms scores are then used to compute the CMF score. The merged term score using the weighted average can be defined as follows:

$$termMerge_{\text{Weighted}}(M_u, t) = \sum_{um_u \in M_u} \omega_{um_u} \cdot um_u(t)$$ (5.32)

Similarly, the merged term score using the maximum value can be defined this way:

$$termMerge_{\text{Max}}(M_u, t) = \max(um_u(t) | um_u \in M_u)$$ (5.33)

Finally, the CMF using the term match approach can be computed as:

$$CMF_{\text{TermMerge}}(u, m) = \frac{\sum_{t \in M} termMerge(M_u, t) \cdot imf(t)}{\sqrt{\sum (termMerge(M_u, t))^2} \cdot \sqrt{\sum imf(t)^2}}$$ (5.34)

5.7 Collaborative-Based Approach

When using a collaborative-based recommender the connections between user and items must be defined. For example, in a typical movie recommender the user and movies (as items) are connected by the ratings of the user.

In an E2SR it is straightforward to use messages as items and connect the users with the messages. Another option is to use the terms as items. Both ideas are discussed next. The main question to be discussed is how to define the weight between the user and the message or between the user and the term.
5.7 Collaborative-Based Approach

5.7.1 Collaboration on Ratings

The simplest way is to connect all users and messages by ratings. Here, a user is connected to a message if a rating exists for the message by the user. The value of connection is then the rating value. However, this does not fulfill the requirement of not using explicit user feedback. Hence, another way of defining the connections and the connection weights is necessary.

5.7.2 Collaboration on Observations

Instead of using explicit user feedback, the observations learned from the features as defined in Equation 5.3 can be used to create an User to Message (UtM) collaborative-based recommender. This is schematically shown in Figure 5.10. Here, users and messages are connected and the connection weight is the observed value. If no observation is made, no connection is created. The prediction by a collaboration algorithm is then based on the interactions of the user.

Figure 5.10: User to Message (UtM) collaboration-based model.

When the user-specific features are computed the Collaboration Match Feature (CbMF) is computed which is just the prediction of the collaboration algorithm collab (see Equation 4.1):

$$CbMF_{UtM}(u, m) = \text{collab}(u, m) \quad (5.35)$$
The result of the \textit{CbMF} is then used in computing the final \textit{relevance score} for the user by the scoring process.

An extension is to use a different collaboration model for each topic. In that case a collaboration model is created separately for each topic. The collaboration models do not influence each other. This way the user behavior of one topic does not influence the recommendation of another topic.

### 5.7.3 Collaboration on Terms

Another idea is to connect users and the extracted terms of the messages as shown in Figure 5.11 to get an \textit{User To Term (UtT)} collaboration-based recommender. The weight of the connection is then the value of the user model $um_u(t)$ (see Equation 5.7). This requires the user model to be learned.

![Figure 5.11: User To Term (UtT) collaboration-based model.](image)

The \textit{UtT CbMF} can then be computed by getting the prediction of the collaboration algorithm for each term of the message and computing the cosine similarity as in the \textit{CMF} (Equation 5.12):

$$CbMF_{UtT}(u,m) = \frac{\sum_{t \in m} \text{collab}(u,t) \cdot \text{imf}(t)}{\sqrt{\sum_{t \in m} \text{collab}(u,t)^2 \cdot \sum_{t \in m} \text{imf}(t)^2}}$$

Again, this can be done either globally for all terms or separately per topic. The topic-specific collaboration model is shown schematically in
5.8 Genetic Algorithm for Optimization

For all algorithms described, many configuration parameters exist. To try all possible configurations would take too much computation power so a Genetic Algorithm (GA) [Gol89] can help to find an optimum configuration.

In the first step all configuration parameters and possible feature values will be defined. Numeric configuration parameters will be mapped to a finite
discrete number of values by defining a minimum and maximum value as well as a precision. For example, a typical feature weight score can take values from 0 to 1. With a precision of 0.1 it comes up to 11 possible values. A boolean feature has two possible values. This way for each configuration parameter the number of possible values is known.

A gen in the genetic algorithm context consists of an instance of feature values for each feature. An example is shown in Figure 5.13. In this example six features are used. The first feature can take five possible values, the second two, the third three and the fourth to sixth two.

In the initial population, a random set of gens is generated and for each gen the fitness is evaluated by computing one evaluation measure. Based on the measure, the best 50% of the population is taken and the crossover operator is applied to them, thereby replacing the other 50%.

The crossover of two gens is based on the number of possible values. Two crossover points are used. The first crossover point is at the first third of the possible feature values, and the second crossover point is at the second third. Then the two gens are mixed by replacing the features between the first and second third, the two new gens are added to the population. After
crossover a mutation is randomly applied. Here, a random feature value is
selected and changed.

The crossover is also shown in Figure 5.13. The sum of all possible values
is 16. The first crossover point is after 5 and 10 possible values, that is after
the first feature and after the third feature. During crossover the feature
fragments are exchanged. Finally in the example the first feature is selected
for mutation for one gen and the value gets changed randomly from 4 to 2.

This process is repeated until a predefined amount of iterations has
been reached or some time has passed. Since the best optimized value is
not known beforehand, it is not possible to use a fixed goal value that the
algorithm should reach.

5.9 Compare Approach to Requirements

In Table 5.3 the requirements of Section 2.9 are checked to see if they are
fulfilled based on the described concepts. The quality requirements are
evaluated in detail in Chapter 7. They will be skipped here.
## 5 Recommender Concept

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Assessment on how fulfilled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filtering of Messages Per User</td>
<td>All approaches allow filtering of messages.</td>
</tr>
<tr>
<td>Recommendation of Top Messages</td>
<td>All approaches allow filtering of messages.</td>
</tr>
<tr>
<td>Without explicit user feedback</td>
<td>All approaches learn only from implicit available features.</td>
</tr>
<tr>
<td>Applicable on E2S</td>
<td>All approaches can be used on E2S. The recommendation process takes a single message as input. Messages of an E2S can be easily forwarded to this process. The topic-specific structure of an E2S is reflect by topic-specific user models.</td>
</tr>
<tr>
<td>Be Adaptive</td>
<td>The short-term approaches are adaptive to changing interests, and the user model adaptation to new terms.</td>
</tr>
<tr>
<td>Incremental Integration</td>
<td>All content-based approaches need an update of the user model which can be updated incremental. The collaborative-based approaches can also be implemented to integrate a change of connections incrementally.</td>
</tr>
<tr>
<td>Recommendation in near Real-Time</td>
<td>For the content-based approaches it comes down to matching the message against the user model which should be a simple computation as long as the size of the user model does not grow infinitely. For collaboration-based approaches it is the prediction of a message for a user, which is also fairly simple.</td>
</tr>
</tbody>
</table>

Table 5.3: Comparison of concept to requirements.
Chapter 6

Realization

In this chapter the architecture for the implementation of the concept (see Chapter 5) is described. This architecture and its implementation are used later for evaluating the concept.

For the implementation an open source framework was created and used as part of this thesis. Open Source Framework for Social Media Stream Recommendation (SPEKTRUM) is described in the Section 6.1. The recommendation process of Section 5.3 including the user model adaptation and short-term approaches is implemented as part of this framework. The scoring and learning implementation are discussed in detail in Section 6.2.

In Section 6.3 it is described which steps are necessary to integrate a E2SA with SPEKTRUM so that the recommendations can be computed and used. As an example the integration of SPEKTRUM into Communote is shown in Section 6.4.

The description of the architecture and technical details will be limited to a level sufficient to understand the evaluation scenario and the argumentation of the thesis.

6.1 SPEKTRUM

SPEKTRUM [Com14c] is a Java-based open source framework founded by Communote GmbH [Com14b] in cooperation with the Chair for Computer Networks at Technische Universität Dresden [Cha14]. This open source framework has been developed as part of the research project with the same name SPEKTRUM. This research is the successor of the research project PRISMA [Com10, Cha12]. The results of PRISMA [Kat+11] have been used for the considerations of the SPEKTRUM project and framework.

1Sächsische Forschungsinitiative zur Personalisierung von Kommunikation und zur Relationsdetektion in Informationsströmen im Unternehmenskontext
2PeRsonalisation of Information StreeM Aggregates
The framework provides the ability to access external source (e.g. RSS or Atom Syndication Format (ASF) feeds) and converts messages into a homogeneous format. As a second functionality it provides a user model learner and a scorer for new incoming messages.

In the next subsection, the main components within SPEKTRUM and the principle of extensible architecture are explained. Afterwards, the data model is described and then (briefly) the persistence and communicator interfaces.

### 6.1.1 Components

Architecturally, one of the main goals of SPEKTRUM is flexibility. The core components (external access, learning, scoring) should not depend on a specific implementation of a third party framework or a specific technology. Therefore, three interfaces are defined:

**Persistence** This interface encapsulates all methods to access, create or modify persisted objects. It is up to an implementation to actually store the objects into a database, a file or somewhere else.

**Communicator** This interface encapsulates the logic to transfer messages to subscribers. An implementation can feel free to send a message via XMPP, Java Messaging Service (JMS) or any other communication protocol.

**Configuration** This interface encapsulates the logic to access the configuration for the components. For example, an implementation of the configuration can load the actual configuration from an XML file or a database.

The interfaces, their implementations and relevant SPEKTRUM components are shown in figure 6.1. SPEKTRUM itself includes some standard implementations, such as an JPAPersistence using the Java Persistence API (JPA) or an XMLConfiguration. The figure also shows that implementations can be provided by other components to use different functionality, such as the Communote Message Queue communicator.

The communicator and persistence interfaces are discussed in Sections 6.1.3 and 6.1.4, respectively.

### 6.1.2 Data Model

The SPEKTRUM Data Model is partially shown in Figure 6.2. Besides the classes given, other classes are also used to model subscriptions, which define the external sources to access. Those classes are omitted here, since they are not relevant for this thesis and further consideration.

The main relevant classes are Message and their dependent ones. Each message can consist of a set of MessageParts that actually hold the content.
Figure 6.1: This UML diagram shows the adapter architecture of a SPEKTRUM component with relevant implementations.
Each MessagePart contains a set of ScoredTerms. A scored term is just a term with a score. The interpretation of the score depends on the usage. For a message part the score defines the certainty of the extracted term. In the context of this thesis no extracted term score is used, and the term score is always 1.

Each message can be part of a MessageGroup. The MessageGroup is a concept to group messages belonging to the same topic. This class represents the topic as described in Section 2.4.3. The user model learner provides an option to learn models independently per message group and therefore per topic.

The class MessageRelation defines a set of messages that belong together for different reasons. One example of a message relation is a discussion, which is a set of messages based on the answer and comment structure. Another example is to express relations based on similar content. SPEKTRUM also defines UserModels consisting of several UserModelEntries which represent the user model term score $um_u(t)$. The term score defines the weight of the term within the user model. An extension is the UserModelEntryTimeBin. This class holds the score of the term for a specific time bin. In that case, the UserModelEntry holds the current value for the existing time bins.

### 6.1.3 Communicator

For performance reasons communication between the components is asynchronous. The Communicator allows the exchange of communication messages such as a message for scoring, a message for an observed interest or a message from an external source. If a message is received by the Communicator it will dispatch them asynchronously to the registered message handler which is able to handle the message.

A simple implementation provided by the framework is the VirtualMachineCommunicator which internally holds a queue in memory and allows communication only within the Java Virtual Machine (JVM). For the Communote integration an implementation is used that is based on JMS.

This communication principle allows for smooth handling of peak amounts of requests as well as to provide the option to distribute the processing load over different application nodes.

### 6.1.4 Persistence

The persistence interface allows the usage of different storage techniques. A JPA persistence and a simple in memory persistence are provided by SPEKTRUM through JPAPersistence and SimplePersistence, respectively. The JPAPersistence stores the data model in a Java Database Connectivity (JDBC) compatible database using EclipseLink [Ecll4]. The SimplePersistence does
Figure 6.2: This UML class diagramm shows the core data elements of the SPEKTRUM component.
not actually store the data but keeps it in memory. This persistence can be used as quick and easy solution for testing and evaluation purposes that do not necessarily need to persist data.

Other possible implementations of the persistence interface is an *Not only SQL* [NoSQL][Cat11] implementation. A comparison of the performance of different persistence technologies could be interesting, but is outside the scope of this thesis.

### 6.2 SPEKTRUM Intelligence

The main components for the SPEKTRUM intelligence are the scorer and the learner. They provide two types of interfaces: The scorer pushes new messages into the scoring process and the learner takes an observation that will start the learning process.

#### 6.2.1 Configuration and Initialization

The intelligence component configurations are nested. For example, the scorer configures and maintains the CMF and CbMF. The UMA configures and maintains a user or topic similarity. Each intelligence component uses its own configuration whereby the configuration of the nested components is also a nested configuration. For example, the scorer configuration maintains a user model adaptation configuration which again maintains a user similarity configuration. The configuration can be set up according to which configuration or variant of an algorithm is used. This allows a flexible configuration of the whole intelligence component including the scoring and learning process.

The initialization of the intelligence follows those three steps:

1. The scorer and learner, and if necessary the user model adaptation and long-term detection, are created and configured.
2. The created components are linked with the communicator. At this point it is possible to pass messages to the scoring process.
3. Jobs are invoked periodically which for example maintain the time-binned user model or compute the user or topic similarity once per day.

#### 6.2.2 Scoring Process

The implementation of the scoring and learning processes is strictly based on the recommender concept (see Section 5.3 and Figure 5.3).

Each process, including the user model adaptation process, follows the *Chain of Responsibility* software design pattern [Gam+95]. Each process step
of Figure 5.3) is mapped to a command which implements the described functionality.

Information Extraction

The information extraction itself is a small process with different commands:

1. **Text Cleaning**: In this step the content of the message is cleaned from Extensible Markup Language (XML) or HTML fragments by removing it with the Jericho library. Also, non-alphabetic and non-numeric characters are replaced by whitespace characters.

2. **Language Detection**: In this step the language of the cleaned text is detected using the framework JLangDetect.

3. **Extracting Tokens**: The cleaned text is tokenized by separating the text into words. The words are stemmed using the detected language and the stemmed tokens are added to the messages as terms. As an alternative, Word or Char N-Grams are computed based on the cleaned text.

4. **Remove Stopwords**: Terms are removed that match a language-dependent stop-word list. The stopword list of Apache Lucene was used.

5. **Count Terms**: The final terms are pushed to a TermFrequency Computer which maintains statistics about how often a term occurs per topic. These statistics are then used to compute the $imf$ which is again used in the CMF.

An elegant process is used to handle topic-specific and language-specific terms: The actual extracted term is augmented by the identifier of the topic (if a topic-specific user model is used) and by the language identifier. Thus a term is unique for different topics and for different languages. For example, the term car of a message in topic with identifier 12 and extracted language en will give the term value: 12#en#car. The hash # itself is excluded as an allowed character for a term. As mentioned before, a topic is represented in SPEKTRUM by the class MessageGroup.

User Model Adaptation

The UserModelAdapter is registered via the communicator as message handler to receive a communication message with the terms and the user to adapt for. The communication message is invoked by the scoring process if the confidence is too low and the terms are unknown. After the adaptation runs and has actually changed the user model, it will invoke the scoring process using the communicator. The relevance score for the user of the changed user model is then recomputed.
The UMA uses either a topic or a user similarity (or both). In both cases the necessary similarity implementations are registered as jobs that will run periodically. The standard configuration is to compute the similarity once every day. During this computation all similarities between all users and between all topics are computed and persisted.

Short-Term and Long-Term User Model

The short-term model maintenance implementation and long-term detection implementation was supported by [Brü13]. Here, a job is registered that will periodically (standard configuration each day) remove old time bins from the short-term user model. Also, per user and per term the occurrences in the time bins are extracted and passed to a long-term detector. This detector will check both for periodic and permanent occurrences. If a long-term has been detected it will be transferred to the long-term model.

Collaboration Match Feature

For the collaboration-based approach the Slope One implementation of Apache Mahout is used in the CbMF. Here, the input values (learning scores for UtM or term scores $u_{m_i}(t)$ for UtT) are transferred linearly from $[0..1]$ to $[-1..1]$. The latter is the standard range for preferences (or ratings) within Apache Mahout. The implementation is not incremental. A whole recomputation is done each day. For a productive usage of the CbMF an incremental implementation has to be used.

6.2.3 Learning Process

The learning process is started upon a new observation. An observation is defined by a message, a user and an interest and is delivered by the communicator. The interest values define the level of interest of the user in the message. The level of interest can be expressed by a value in the interval $[0..1]$ with 0 representing the lowest and 1 the highest interest. An observation can originate from:

1. An explicit rating of the user.
2. A new message is identified as interesting for the user by the scoring process.
3. An interaction of the user within the system (e.g. the user liked the message).

After the observation has started, the learning process loads the related messages. It is then being determined if an information extraction has already run and the terms are available. If the associated message has run
through the scoring process, the terms are available. If the message is based on an older message which has not been scored before then the information extraction must run here.

Finally, if all information are available, the user model is updated. Here the associated user model term score is updated with each of the extracted terms of the message. If the time bins are maintained they are updated as well. There are different possible update strategies for the user model. The main ones used are to count the number of occurrences of the term as well as the sum of the interest values.

Multiple Observations

Observations can be inferred from interactions of the user with the message which will change the user model. Later, the user may directly interact with the message by liking it or by submitting an explicit rating. In the latter case the explicit rating should override the inferred one. To achieve this, the old implicit learning is learned back in the user model before integrating the new message. Learned back for the term count user model learning strategy is simple: the count and sum of the user model term score is decremented by the old observation. For the incremental user model learning strategy the old observation is treated as a negative observation. When learning back on an incremental user model: if at the current state the old observation would increment the term score then the term score will be decremented by the old observations’ interest. If it would decrement the term score then the term score will be incremented when learning it back.

6.3 Integration of SPEKTRUM into Applications

In this section, how to connect SPEKTRUM to an existing application is discussed. This includes the steps which are necessary to integrate recommendations in an application.

The first step is to define the basic configuration. This includes to choose the type of database where SPEKTRUM persists data in, as well as defining the scorer and learner configurations for the intelligence component. A standard communicator can be used that keeps the communication messages in memory or specific implementation can be provided.

The second step is to implement a mechanism in the application to push a new message to the communicator which will then take care of invoking the scorer and the scoring process. At the end of the process the relevance scores per user for the message are stored in the SPEKTRUM database.

The third step is to access the relevance scores for filtering. One option is to access the SPEKTRUM database by the application. Another option is to add a process step at the end of the scoring process that will push the
computed relevance scores back to the application. In that case the application is more flexible in storing the relevance score in an appropriate format for faster access, e.g. using database joins or its own caching mechanism.

The last step is to register and run the periodical jobs for similarity computation and short-term model maintenance. SPEKTRUM provides a way to directly use native Java Timer. As an alternative, if the application already uses its own scheduling pool, it can be integrated to use the same pool.

6.4 Communote

For evaluation, Communote [Com14] is used and therefore SPEKTRUM was integrated with Communote. Communote is a social media system for the enterprise developed by the Communote GmbH as already described in Section 2.4.1. It allows to post messages organized in topics, share messages with other users and define access levels on topics. In this section the integration is described in detail.

6.4.1 Communote Architecture

Communote is implemented in Java stack using Hibernate[Bo14] and the Spring Framework[Spr14] as application framework. It maintains an Open Services Gateway initiative (OSGI) plugin layer that allows implementation of further extensions.

Communote provides a Representational State Transfer (REST) API to manage messages or topics. It also maintains an embedded message queue that can be used to send messages to or receive messages from Communote.

6.4.2 Communote Integration

The main components of the Communote integration are shown in Figure 6.3. SPEKTRUM’s flexible architecture allows an easy integration into existing systems.

For bringing Communote and SPEKTRUM together two plugins exist: ExStream and MyStream. ExStream manages the access to external sources and is not of further interest in this thesis. The MyStream Plugin provides front-end functionality to Communote, e.g. for rating messages or filtering for relevant messages. MyStream also converts the messages between the SPEKTRUM and Communote data models and MyStream manages the learning process and invokes the scoring process.

---

3 [http://docs.oracle.com/javase/7/docs/api/java/util/Timer.html](http://docs.oracle.com/javase/7/docs/api/java/util/Timer.html)
4 The Ex stands for External.
6.4 Communote

For accessing the Communote services, the MyStreamPlugin mainly uses the message queue as well as the plugin functionality. The latter allows to invoke methods directly within the core of Communote.

6.4.3 Communote Data Model

The core data model of Communote consists of Notes, Topics and Tags. Notes correspond to messages and topics correspond to message groups. Topics within Communote have a set of assigned access rights. The access rights are not available within the SPEKTRUM data model. However, during the scoring process a callback is used to determine the available users for a message. Besides that there are direct messages which are messages that are only visible for a specific set of users.

Notes can have a threaded structure which forms if an author answers or comments on an existing note. This thread is called a discussion and corresponds to the MessageRelation of SPEKTRUM.

All notes can be bookmarked or liked. An author can mention another user by including the login in the text of the note or by explicitly selecting the user. These interactions can be used as further input for the learner.
6 Realization

Figure 6.4: Communote rating front-end implementation: This shows the extension of the Communote front-end by the MyStream plugin. On the right side the score filter allows the filtering of messages by the computed relevance score and for each message the user can rate a message as positive (relevant) or negative (irrelevant). Also, the computed relevance score itself is shown.

6.4.4 MyStream Communote Plugin

The purpose of the MyStream Communote Plugin is to integrate the functionality of SPEKTRUM and map it to the logic of Communote. The plugin implements the Communicator interface which uses a JMS message queue provided by Communote. As a JMS provider, ActiveMQ[^14a] is used.

The plugin registers a listener in Communote, so that the plugin receives all notes that are stored within Communote. It then creates a SPEKTRUM message with the message group being the topic of the note. Afterwards, the message is injected into the scoring process.

The plugin changes the scoring process by adding a new processing step to the end. In this step the message relevance scores are stored within Communote so that the score is available for each message. The score is then used for filtering and for visualization.

The plugin observes events so that it gets notified once a user explicitly rates a message. Subsequently, the plugin creates an observation and invokes the learning process of SPEKTRUM.

During the scoring process a callback is used to determine the users who have access to a given message group. In the MyStream Communote Plugin this is implemented as a callback to return all the users having at least a read right for the topic (based on the message group).

6.4.5 Front-End and Interaction

Within the front-end of Communote the MyStream Plugin allows to rate a message as interesting or non-interesting as shown in Figure 6.4. In both cases the learning process will be invoked on a change of interest.

The computed score is shown next to the options of rating the message. The plugin also adds a score-filter widget to Communote as shown in Figure 6.4. Here, the user can choose a score and the message stream is filtered to include only messages that have a higher relevance score than the selected one.
6.4.6 Initial User Model Learning

After the first deployment of the plugin within Communote no initial user model is available. To overcome this cold start problem the plugin adds an administrative functionality which allows the administrator to start an initial bulk learning. Here a set of users can be specified and the messages within a certain time span are injected into the scoring process. As stated in Section 6.2 the scoring process itself invokes the learner. The result is an initial user model for the defined users.
Chapter 7

Evaluation

This chapter describes the evaluation setup, the evaluation strategy and the measures evaluation is based on. The goal of the evaluation is to evaluate the approaches developed in concept (Chapter 5) and to show which approaches help to find relevant messages. First, in Section 7.1 it is described what exactly to evaluate to fit the requirements defined in Section 2.5. Then, in Section 7.2 the dataset used for evaluation is described and analyzed before in Section 7.3 the measures to compare different configurations are described and developed. In Section 7.4 the implementation of the evaluation is briefly described.

In Section 7.5 the basic algorithm is optimized and a baseline is defined which is used for comparison. The optimization of the scoring feature weights is described in Section 7.6. The optimized basic algorithm and optimized scoring feature weights is the basic configuration used for all other runs of the evaluation.

The optimization and evaluation of the content-based approach is described in Section 7.7. This is followed by the user model adaptation approach evaluated in Section 7.8 and the short-term approach in Section 7.9. In Section 7.10 the collaboration-based approach is evaluated. Finally, in Section 7.11 all evaluation runs are compared.

### 7.1 Evaluation and Requirements

The main goal of the evaluation is to evaluate the quality of the algorithm and the different approaches. The quality is defined either through the number of relevant messages identified or the sorting of the results with the relevant message on top. This is closely related to the requirements defined in Section 2.9. The first two requirements are important to consider for the evaluation:

1. Filtering of Messages Per User
2. Recommendation of Top Messages per Day and Week

In order to filter messages, all messages must be considered. The quality measure includes the number of relevant messages found. For the recommendation of the top messages the quality is defined through the top-n relevant messages per day or week. Also, only the messages the user did not interact with should be considered. As defined in the requirements (see Section 2.9.2) there are two options: message-based and discussion-based interaction exclusion.

In both cases, when the user interacted with the message or discussion, respectively, there is no need to recommend it again. This exclusion can be computed based on the features defined. For the message-based exclusion this applies to the Author Feature and Mention Feature. In the case of the discussion-based exclusion this applies also to the Author, Mention, Discussion Mention and Discussion Participation Features.

A rating refers to exactly one user and exactly one message and states the relevance of the user to the message. Based on the exclusion, the following rating sets can be distinguished:

**All Ratings (Rall)** Includes all ratings.

**Ratings with Interaction (RwI)** Includes all ratings where the rating user is the author or is mentioned in the message of the rating. The Author Feature or User Feature for the user and message is present.

**Ratings with Discussion Interaction (RwDI)** Includes all ratings where the rating user participated or is mentioned in the discussion of the message. RwDI does not contain ratings of RwI: RwI ∩ RwDI = ∅.

**Ratings without Interaction (RwoI)** Includes all ratings except ratings with interactions.

**Ratings without Discussion Interaction (RwoDI)** Includes all ratings except ratings with interaction of discussions. RwoDI is a subset of RwI: RwDI ⊂ RwoI. Also, RwoI is the union of RwoDI and RwI: RwoDI ∪ RwDI = RwoI.

In Figure 7.1 all rating sets and their relationship are visualized. RwoI and RwI will give exactly Rall:

\[
RwoI ∪ RwI = Rall \quad (7.1)
\]
\[
RwoI ∩ RwI = ∅ \quad (7.2)
\]

Also, RwoDI, RwDI and RwI will give exactly Rall:

\[
RwoDI ∪ RwDI ∪ RwI = Rall \quad (7.3)
\]
\[
RwoDI ∩ RwDI ∩ RwI = ∅ \quad (7.4)
\]
7.2 Dataset for Evaluation

In order to the requirements three rating sets are relevant. Rall is used for the requirement of filtering all messages because this rating set uses all ratings without excluding all messages. For identifying the top-n messages, messages with interaction should be excluded. The exclusion can be either based only on the message or only on the discussion. In the first case the RwoI is representing the message exclusion, and in the second case RwoDI is representing the discussion exclusion. The discussion exclusion is the most restrictive one.

Figure 7.1: Rating sets: All Ratings (Rall) contains Ratings with Interaction (RwI) and Ratings without Interaction (RwoI). Ratings without Interaction (RwoI) can be separated into Ratings without Discussion Interaction (RwoDI) and Ratings with Discussion Interaction (RwDI).

7.2 Dataset for Evaluation

The dataset used for evaluation has been obtained using the E2SA Communote [Com14b] deployed at Communardo Software GmbH [Com14a]. Overall, over 200,000 messages from more than 100 users have been created within this system. During the evaluation period 10 employees rated messages between December 2012 and April 2013. For each message a user was able to rate a message as either positive (relevant), negative (irrelevant) or as unidentified. Unidentified ratings are not considered further. For a relevant rating a goal relevance score of 1 is used and for an irrelevant rating a relevance score of 0. The rating was integrated directly into Communote as shown in the implementation (see Figure 6.4 and Section 6.4.5). The users were free to rate or not to rate a message, and not all messages during that time span were rated. A Relevance Manifest has been presented to each user who rated. The goal of this manifest is to give a common understanding of what relevance in the context of this thesis is. The manifest is shown in Appendix. In total 30,123 ratings were submitted by 10 users and the ratings ranged over 14,650 unique messages. A total of 23,105 message were created during the evaluation time span.

In Table 7.1 the number of positive (relevant) and negative (irrelevant) submitted ratings per rating set is presented. The majority of ratings are
7 Evaluation

<table>
<thead>
<tr>
<th>All</th>
<th>Ratio of Positive</th>
<th>F₁</th>
<th>F₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>5702</td>
<td>19%</td>
<td>0.318</td>
<td>0.539</td>
</tr>
<tr>
<td>2518</td>
<td>87%</td>
<td>0.931</td>
<td>0.971</td>
</tr>
<tr>
<td>556</td>
<td>42%</td>
<td>0.590</td>
<td>0.783</td>
</tr>
<tr>
<td>2628</td>
<td>10%</td>
<td>0.184</td>
<td>0.361</td>
</tr>
<tr>
<td>3184</td>
<td>12%</td>
<td>0.209</td>
<td>0.398</td>
</tr>
</tbody>
</table>

Table 7.1: Number of positive (relevant), negative (irrelevant) and all ratings for each rating set of the evaluation dataset. The sum of \( R_{all} \) is combined of \( R_{wl} \) and \( R_{wo} \). The ratio of positive gives the percentage of positive ratings to all ratings. If a recommender will retrieve all messages as relevant the \( F_1 \)-Score (\( F_1 \)) and \( F_2 \)-Score (\( F_2 \)) values shown will be measured. Those value should show a lower boundary for a useful recommender algorithm. Both measures are described in detail in Section 7.3.2.

The number of ratings is concentrated in the \( R_{wl} \) and \( R_{wo} \) rating set. The number of positive ratings shows that for messages with direct interaction an overwhelming amount (87%) are positive ratings. For \( R_{wo} \) only 12% are positive ratings and for \( R_{wo} \) only 10% are positive ratings. This indicates that the hardest task will be to identify relevant message for \( R_{wo} \).

The number of ratings per user ranges differently as shown in Figure 7.2. Five users submitted more than 1,000 ratings each and the other five only rated sporadically. In Figure 7.3 the distribution of the ratings per user for each month is shown. Users A to D rated messages in more than one month. The majority of ratings were submitted in January and February 2013.

In Figure 7.4 the number of positive and negative ratings for both \( R_{wl} \) and \( R_{wo} \) are shown for each user. For all users there are more positive ratings than negative for \( R_{wl} \), and for 7 users (A-G) more negative ratings clearly exist for \( R_{wo} \).

In Figures 7.5 to 7.7 the number of ratings are shown per message for the creation date of the messages. Each bar shows the ratings from one day. Since the users were encouraged to rate as soon as possible after a message was created, the rating date and creation date can be assumed to be similar. Here, it is shown that most ratings are concentrated between January and February 2013, and there are also some ratings for March 2013. In Figure 7.6 only the \( R_{wl} \) are shown. Again, it can be observed that most of the ratings are positive. In contrast, in Figure 7.7 the \( R_{wo} \) per day are shown where the most ratings are negative and only a few have been rated positive.
7.2 Dataset for Evaluation

![Bar Chart]

Figure 7.2: Number of ratings for each user with absolute values.
Figure 7.3: Rating distribution per user for each month.
### 7.2 Dataset for Evaluation

<table>
<thead>
<tr>
<th>User</th>
<th>Positive RwI</th>
<th>Negative RwI</th>
<th>Positive RwoI</th>
<th>Negative RwoI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>807</td>
<td>162</td>
<td>1,159</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9,947</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>298</td>
<td>103</td>
<td>409</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5,774</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>256</td>
<td>2</td>
<td>736</td>
<td>3,633</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>806</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>113</td>
<td>64</td>
<td>410</td>
<td>2,779</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1,276</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>170</td>
<td>5</td>
<td>228</td>
<td>415</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>144</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td>17</td>
<td>33</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>J</td>
<td></td>
<td></td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 7.4: Rating distribution per user for Ratings with Interaction (RwI) and Ratings without Interaction (RwoI).
Figure 7.5: Number of ratings per day for all messages. Each x-axis tick represent the start of a week.

Figure 7.6: Number of Ratings with Interaction (RwI) for each day. Each x-axis tick represent the start of a week.
7.3 Measures for Evaluation

Before comparing different evaluation runs the measures for comparison have to be selected carefully. The measure to use should fit the requirements defined, and if possible give a good interpretation on the performance. In this section, four measures will be introduced:

- **$F_1$-Score ($F_1$)** This measure is evaluated against the whole dataset and best used to evaluate the Filtering of Messages Per User requirement.

- **$F_2$-Score ($F_2$)** This measure is evaluated against the whole dataset and best used to evaluate the Filtering of Messages Per User requirement. In contrast to $F_1$, it focuses more on finding the correct messages instead of being precise.

- **Time-Binned Mean-Average-Precision ($TB$-MAP)** This measure is evaluated on the top elements of each day or week separately. It also takes the ordering of the elements into account and is best used to evaluate the Recommendation of Top Messages per Day and Week requirement.

- **Time-Binned Precision@ ($TB$-P@$k$)** This measure is similar to $TB$-MAP but it provides a better understanding of the number of messages found per time bin. It reflects the percentage of found relevant messages to the number of possible messages in the top-$n$ per day or week.
7 Evaluation

7.3.1 Basic Definitions

First, some mathematical definitions are presented to further develop the measures. A relevance score and a set of scores can be defined as a tuple (or a set of tuples) containing a score \( s \) for a user \( u \) for a message \( m \):

\[
\text{score} = (u, m, s), s \in [0..1], u \in U, m \in M
\]  \hspace{1cm} (7.5)

\[
\text{scores} = \{(u_1, m_1, s_1), (u_2, m_2, s_2) \ldots (u_n, m_n, s_c)\}
\]  \hspace{1cm} (7.6)

The user-submitted ratings themselves can be represented using this notation by using a score of either 1 or 0 for a relevant or irrelevant rating, respectively:

\[
\text{ratings} = \{(u_1, m_1, r_1), (u_2, m_2, r_2) \ldots (u_n, m_n, r_n)|\text{user}\ u_i\ \text{submitted a rating}\ r_i\ \text{for message}\ m_i\}
\]  \hspace{1cm} (7.7)

The EvalScores are the computed scores where a rating exists:

\[
\text{evalScores} = \{(u, m, s)|\exists r(u, m, r) \in \text{ratings} \land (u, m, s) \in \text{scores}\}
\]  \hspace{1cm} (7.8)

Since the ratings are globally defined for an evaluation and for better understanding, the ratings parameter is omitted in the further definitions.

To compute the precision and recall, a decision has to be made as to whether the computed score is relevant or not. The function \( \text{threshold} \) limits the elements of a set of scores to those satisfying a score threshold:

\[
\text{threshold}(\text{scores}, \text{th}) = \{(u, m, s)|s \geq \text{th} \land (u, m, s) \in \text{scores}\}
\]  \hspace{1cm} (7.9)

With this definition, the scores retrieved as relevant respecting a threshold \( \text{th} \) can be defined as:

\[
\text{relevant}_{\text{th}}(\text{scores}) = \{(u, m)|s \geq \text{th} \land (u, m, s) \in \text{scores}\}
\]  \hspace{1cm} (7.10)

Also with this notation, all relevant user submitted ratings can be formally expressed as:

\[
\text{relevant}_{\text{th}}(\text{ratings}) = \text{threshold}(\text{ratings}, 1)
\]  \hspace{1cm} (7.11)

Another notation to be used is \( |_n \) which will limit a set of scores to the top-\( n \) scores in the set:

\[
\text{scores}|_n := \{(u, m, x)|(u, m, x) \in \text{scores} \land |\text{threshold}(TB, x)| \leq n\}
\]  \hspace{1cm} (7.12)

Later, it is necessary to identify relevant ratings based on a subset of scores, e.g. per user or for a certain time bin or the relevant ratings of the evaluation dataset. Here, the notation \( \text{ratings}_{\text{scores}} \) refers to all ratings where a computed score exists in \( \text{scores} \). In difference to \( \text{evalScores} \), it contains the scores with ratings and computed scores. This works only if the scores are computed for all message-user combinations for which a rating exists:

\[
\text{ratings}_{\text{scores}} = \{(u, m, r)|(u, m, r) \in \text{ratings} \land \exists s(u, m, s) \in \text{scores}\}
\]  \hspace{1cm} (7.13)
7.3 Measures for Evaluation

7.3.2 Basic Measures

Common measures for information retrieval are Precision ($P$), Recall ($R$) and F-Scores. In general those measures are defined as:

$$P = \frac{\text{Number of correctly retrieved items}}{\text{Number of relevant retrieved items}}$$  \hspace{1cm} (7.14)

$$R = \frac{\text{Number of correctly retrieved items}}{\text{Number of existing relevant items}}$$  \hspace{1cm} (7.15)

$$F_\beta = \left(1 + \beta^2 \right) \cdot \frac{P \cdot R}{\beta^2 \cdot P + R}$$ \hspace{1cm} (7.16)

For the $F$-Score the factor $\beta$ defines how to weight the precision and the recall. For $\beta = 1$ the $F_1$-Score is the harmonic mean between precision and recall, weighting both equally. In contrast, $F_2$ gives more impact to the recall. Therefore, a higher $F_2$ is achieved if more correct messages are found while also finding more incorrect messages. $F_1$ focuses on the mean between being correct and precise.

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$ \hspace{1cm} (7.17)

$$F_2 = 5 \cdot \frac{P \cdot R}{4 \cdot P + R}$$ \hspace{1cm} (7.18)

All three measures lead to value in the interval $[0..1]$. Based on the basic definitions described before, precision $P$ and recall $R$ can be can be expressed as follows. Here, relevant ratings and relevant scores are the retrieved relevant results:

$$P_{th}(scores) = \frac{|\text{relevant}_1(ratings) \cap \text{relevant}_{th}(scores)|}{|\text{relevant}_{th}(scores)|}$$ \hspace{1cm} (7.19)

$$R_{th}(scores) = \frac{|\text{relevant}_1(ratings) \cap \text{relevant}_{th}(scores)|}{|\text{relevant}_1(ratings)|}$$ \hspace{1cm} (7.20)

The precision determines how many of the returned results are correct and the recall determines how many of all the available relevant results have been found. Both measures are dependent on each other. It is simple to achieve a recall of 1 by returning all results as relevant. However, in this case the precision drops. This is the reason why $F$-Scores are used to combine both.

All three of the measures are evaluated on the whole dataset. If the recommender returns a computed relevance score per message a decision has to be made as to which value the message is returned as relevant. With the definition of Equation 7.9 and $th$ as threshold $P_{th}$ is the precision with scores $\geq th$ retrieved as relevant. The same applies for $R_{th}$, $F_{1,th}$ and $F_{2,th}$.
7 Evaluation

Now, to select a measure for comparison either a fixed threshold can be used, or the maximum measure for a threshold is taken:

\[ \text{Max-}F_1(\text{scores}) = \max\{F_{1,th}(\text{threshold}(\text{scores}, th)) | th \in [0..1]\} \]  \hspace{1cm} (7.21)

The \text{Max-}F_1 and \text{Max-}F_2 measures are determined as follows:

1. Start with a threshold \( th = 0 \) and then increment it in iterations by a step value. (In the evaluation incremental steps of 0.001 are used.)
2. Compute the precision, recall, \( F_1 \) and \( F_2 \) scores for each step.
3. Determine the maximum \( F_1 \) and \( F_2 \) scores of all steps.

From this point forwards \( F_1 \) and \( F_2 \) will be used as short for \text{Max-}F_1 and \text{Max-}F_2, respectively.

**Precision@**

The \( F_1 \) and \( F_2 \) scores are computed on the whole dataset. It gives an impression of the overall performance of a recommender algorithm to find all relevant items. This is the requirement \text{Filtering of Messages Per User} defined in Section 2.9.1.

In contrast, there is the requirement \text{Recommendation of Top Messages per Day and Week} defined in Section 2.9.2. Here, the recommender is required to find a specific number of the relevant messages. For example, if there is a high number of relevant messages no user will go through a list of all found messages. Instead, the user wants to get only 5 or 10 messages, preferably relevant messages, such as in typical web search scenarios. To reflect this requirement the measure \text{Precision@}k and \text{Average-Precision@}k can be used. The \( k \) stands for the top \( k \) items to be considered. Instead of evaluating the measure on the whole dataset only the top \( k \) elements are considered. The \text{Precision@}k \( (P@k) \) is then defined as:

\[ P@k = \frac{\text{Number of correct retrieved items in the top } k \text{ elements}}{k} \]  \hspace{1cm} (7.22)

Based on the previous definitions, the top \( k \) items of the recommender are those with the highest score expressed by \( \text{scores}|_k \) (see Equation 7.12). There might be cases where the number of existing relevant items is less than \( k \). A recommender will not be able to return \( k \) relevant elements and then not be able to reach a \( P@k \) of 1. Later, different \( P@k \) will be combined and it is therefore crucial that \( P@k \) is defined properly. If less than \( k \) relevant items exist, the number of existing relevant items \( \text{relevant}_1(\text{ratings}\text{scores}) \) will be used instead of \( k \). The \( P@k \) can then be defined as:

\[ P@k(\text{scores}) = \frac{|\text{relevant}_1(\text{ratings}\text{scores}) \cap \text{scores}|_k|}{\min(k, |\text{relevant}_1(\text{ratings}\text{scores})|)} \]  \hspace{1cm} (7.23)
Average Precision @

The $P@k$ does not care about the ordering of the items. It does not make a difference whether a relevant item is returned first or at the $k$-th position. The Average-Precision@$k$ is taking the ordering of the elements into account. With $relevant(i)$ returning 1 if the item at the $i$-th position is relevant, Average-Precision@$k$ ($AP@k$) is defined as:

$$AP@k = \frac{\sum_{i=0}^{k} relevant(i) P@i}{k}$$  \hspace{1cm} (7.24)

Let $rank(score, i)$ return the element with $i$-th highest score:

$$rank(score, i) = (u, m) \text{with } i\text{-th highest } s \land (u, m, s) \in scores$$ \hspace{1cm} (7.25)

With $rank(score, i)$ the function $relevant(scores, i)$ can be defined as follows:

$$relevant(scores, i) = \begin{cases} 
1 & rank(score, i) \in relevant(scores) \\
0 & \text{otherwise}
\end{cases}$$ \hspace{1cm} (7.26)

With those definitions, $AP@k$ based on the computed scores can be defined as:

$$AP@k(scores) = \frac{\sum_{i=0}^{k} relevant(scores, i) P@i}{\min(k, |relevant(scores)|)}$$  \hspace{1cm} (7.27)

Again, as in Equation 7.23 for the $P@k$, the number is limited by the minimum of either $k$ or the number of available relevant ratings in the evaluation dataset.

Mean Average Precision

As stated, when computing the $P@k$ or $AP@k$, the ordering of the elements is relevant. To reflect the scenario, the ordering must be user based, since the messages will be presented per user. This means $P@k$ and $AP@k$ must be executed and averaged per user. This is commonly known as Mean-Average-Precision. With $U_{rel}$ as a set of users with at least one relevant rating, the Mean-Average-Precision ($MAP$) of the scores can be computed as follows:

$$MAP@k(scores) = \sum_{u \in U_{rel}} \frac{AP@k(scores_u)}{|U_{rel}|}$$  \hspace{1cm} (7.28)

For users that do not have a relevant rating the $AP@k$ would always be 0 and it would falsify the result. Therefore these users are omitted from the measures.
7 Evaluation

7.3.3 Time-Binned Measures

In Section 2.9 the requirement Recommendation of Top Messages per Day and Week is formulated. The measures described before are evaluated over all messages and are not useful for measuring the quality of the requirement. This is the reason for introducing time-binned measures. Instead of evaluating measures $P_{@k}$, $AP_{@k}$ and $MAP$ over all ratings in the evaluation dataset, the dataset will be split into time bins and measures are computed independently per time bin. The computation for the time-binned measure follows this procedure:

1. Compute scores for each user and message.
2. Compute evalScores by filtering scores for the evaluation data available. For example, if no rating in the evaluation dataset is available for a message $m$ for a user $u$ it will be ignored in the following steps.
3. Split the messages into time bins of equal length (e.g. per day or per week). Each message occurs in exactly one time bin.
4. Compute the measure per time bin.
5. Aggregate the measures per time bin. Only consider time bins where relevant ratings exist.

Two time-binned measures are used for comparison: Time-Binned Precision@ $\langle TB-P_{@k} \rangle$ and Time-Binned Mean-Average-Precision $\langle TB-MAP \rangle$. The first one gives a good, understandable indication of how many relevant results have actually been found and the second for comparing different recommendation algorithms.

Definitions

The $TB$ function splits a set of scores into time bins based on the creation date of the message. $TB$ stands for TimeBin. The index of the function defines the length of a time bin, e.g. Month where each time bin contains scores for one month:

$$TB_{\text{Month}}(scores) = \{ (u_1, m_1, s_1), (u_2, m_2, s_2), \ldots \}$$
$$\{ (u_3, m_3, s_3), \ldots \}$$
$$\{ \}$$
$$\vdots$$
$$\}$$

(7.29)

To access a single time bin the following notation is defined, assuming time-ordered time bins:

$$TB_i = scores_i : \text{Is the } i\text{-th time bin.}$$

(7.30)
The function $hasRelevant$ is used to determine if the set of computed scores contains at least one relevant message:

$$hasRelevant(scores) = \begin{cases} 1 & |relevant_1(ratings_{score})| > 0 \\ 0 & \text{otherwise} \end{cases} \quad (7.31)$$

This definition can only be applied if $scores$ contains all possible message user pairs.

**Time-Binned Precision@**

The Time-Binned Precision@ is based on the $P_k$. Here, the number of all successful relevant retrieved messages in the top-$k$ messages is computed. Then, the number of existing relevant messages is determined. The ratio of both gives the definition for $TB-P@k$

$$TB-P@k(evalScores) = \frac{\sum_{s_{tb} \in TB(evalScores)} |relevant_1(ratings_{s_{tb}}) \cap s_{tb}|k|}{\sum_{s_{tb} \in TB(evalScores)} \min(k, relevant_1(ratings_{s_{tb}}))} \quad (7.32)$$

This measure allows a simple interpretation of the quality performance of a recommender algorithm. E.g. a value of 0.7 means that 70% of all possible relevant messages have been found, with respect to top-$k$ messages per time bin.

**Time-Binned Mean Average Precision@**

The $TB-P@k$ does not respect the ordering of the elements or a user differentiation. For a more precision comparison of recommenders a measure Time-Binned Mean-Average-Precision can be defined that is based on $MAP$.

Let $\omega$ be the number of time bins that contain at least one relevant message:

$$\omega = \sum_{scores_{tb} \in TB(evalScores)} hasRelevant(scores_{tb}) \quad (7.33)$$

With this definition the Time-Binned Mean-Average-Precision can be computed as follows:

$$TB-MAP@k(evalScores) = \frac{1}{\omega} \sum_{scores_{tb} \in TB(evalScores)} MAP@k(scores_{tb}) \quad (7.34)$$

This can be used with the definition of Equation 7.28 to get:

$$TB-MAP@k(evalScores) = \frac{1}{\omega} \sum_{s_{tb} \in TB(evalScores)} \sum_{u \in U_{tb,rel}} \frac{AP@k(s_{u,tb})}{|U_{tb,rel}|} \quad (7.35)$$

This measure equally aggregates the $MAP$ of each time bin to one measure.
7 Evaluation

7.4 Implementation of Evaluation

The main evaluation classes are shown in Figure 7.8. The evaluation implementation uses the SPEKTRUM Framework as described in Chapter 6. The EvaluationExecutor uses an EvaluationConfiguration to configure the Scorer and Learner of the Spektrum Framework. Then, for each message the scoring and learning process is invoked as follows:

1. The oldest yet to be processed message is sent to the scoring process.
2. If invoked, the learning and user model adaption process runs until completion for this message.
3. If test ratings are available for this message the current computed relevance score for the message and the target score (rating) is sent to the Evaluator.

The Evaluator just takes the computed and target relevance scores and stores them in a ComputedRanks file for each evaluation run. The Evaluation Executor Starter has a set of EvaluationConfigurations, each holding all the necessary configurations for the scorer, learner and the evaluation itself (e.g. how to use ratings for training). The starter then just takes the configuration and starts a new evaluation run using the Evaluation Executor. After each evaluation run the Measure Computer takes the ComputedRanks file and computes the necessary measures, such as precision, recall, F-Scores, the time-binned measures and stores them in a new file PrecisionRecall.

7.5 Basic Algorithm Optimization

As described earlier, there are many configuration parameters for the algorithm. In order to find suitable configuration parameters they will be optimized using the genetic algorithm as described in Section 5.8.

For optimization and evaluation the dataset is split. Since the dataset is unbalanced for different users the splitting is done for each user separately and combined afterwards. This way, each partition contains approximately the same ratio of ratings per user which avoids the possibility that one dataset contains no ratings for a specific user.

For optimization, 30% of the dataset is used to find the best parameters and the other 70% is used as test data to evaluate against the measures.

7.5.1 Configuration

In the first step the information extraction and content match configuration parameters are optimized. In the following, the optimized parameters are used for all further configurations and they are not changed.
Information Extraction

There are several options that can be applied to the information extraction process. Those options are shown in Table 7.2.

Content Match Feature Configuration

The possible values for the content match configuration are shown in Table 7.2. The variety in this configuration will give confidence for choosing a useful term match similarity. The Term Weight Strategy and the Term Vector Similarity Strategy are used for matching the message of the terms against the user model. The maximum vector similarity just takes the maximum value out of the matching user model entries. Average takes the weighted average of the user model entries. The trivial term weight just uses a weight of 1 for each term. The Inverse Message Frequency (\(imf\)) term weight is defined in Equation 5.11 (see Section 5.4.2). If the option Treat Missing User Model...
Evaluation

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Description</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Stemming</td>
<td>Use stemmed single words</td>
<td>true, false</td>
</tr>
<tr>
<td>Use Word N-Grams</td>
<td>Use word N-grams</td>
<td>true, false</td>
</tr>
<tr>
<td>Use Char N-Grams</td>
<td>Use char N-grams</td>
<td>true, false</td>
</tr>
<tr>
<td>N-Grams Length</td>
<td>The length of n-grams</td>
<td>1, 2, ..., 10</td>
</tr>
<tr>
<td>Do Tags</td>
<td>Use the tags of the message for extraction</td>
<td>true, false</td>
</tr>
<tr>
<td>Do Tokens</td>
<td>Extract and use tokens out of the messages content</td>
<td>true, false</td>
</tr>
<tr>
<td>Add Tags To Text</td>
<td>Tags of the messages are added to the text and</td>
<td>true, false</td>
</tr>
</tbody>
</table>

Table 7.2: Configuration options for information extraction.

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Description</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term Weight Strategy</td>
<td>Defines how to determine term weights.</td>
<td>Trivial or Inverse Message Frequency ((\text{imf})).</td>
</tr>
<tr>
<td>Term Vector Similarity Strategy</td>
<td>Defines how to match the terms of a message against the user model.</td>
<td>Maximum, Average or Cosine-Similarity.</td>
</tr>
<tr>
<td>Treat Missing User Model Entries As Zero</td>
<td>Defines how to deal with missing terms in the user model.</td>
<td>true, false</td>
</tr>
</tbody>
</table>

Table 7.3: Configuration options for the Content Match Feature (CMF).

Entries As Zero is activated, a nonexistent term score in the user model for a term of the message is treated as 0. If the option is deactivated, only existing term scores in the user model are used for computing the CMF. Hence, if there are many unknown terms in the user model, the resulting CMF score will be lower if the option is used in contrast to ignoring the missing values.

Bootstrapping Problem

There is a bootstrapping problem, because before finding the optimized parameters other parameters that have not been optimized must be used. One option would be to optimize all parameters together but this would prohibit comparison between different approaches.
For this reason, in this first optimization fixed feature weights and learning feature weights are used to optimize the information extraction and CMF configuration. The Author Feature is used with a score of $f_{\text{author}} = l f_{\text{author}} = 1$ as scoring and learning feature weight.

**Optimized Configuration**

As mentioned, 30% of the ratings of the dataset have been used for optimization. A population of 100 different configurations has been generated randomly. This population was used by the genetic algorithm and for each configuration the $F_1$ measure has been computed. 50% of the configurations with the highest fitness in the population is used for the next population. The other 50% are new created configurations by the genetic operations crossover and mutation (as described in Section 5.8). Since there is no goal measure value to be reached, there is no hard termination condition to stop the optimization. The termination condition at least 100 iterations must ran or at least 8 hours of computation time must be exceeded. Finally, the latest population can have two states:

1. Most of the configurations have computed measure values that are close to each other and their configurations are similar (only one parameter is slightly different).
2. There are (completely) different configurations with similar measures.

In the second case the optimization process was continued for another 100 iterations and at least 8 hours of computation. If once again the second case is reached it is an indication of unstable parameters: a small change leads to different results. Even if the first case is reached finding the optimal configuration parameters is not guaranteed, only one that is a local maximum.

When optimizing the information extraction and CMF configuration, the population reached the first case after the run. This configuration is shown in the Table 7.4. In this optimization configuration there are the following interesting points to notice:

1. Word N-Grams wins against Char N-Grams and pure stemming.
2. Very surprisingly, using the tags of a message does not have any impact; the best configuration found does not use tags.
3. The combination of using inverse term frequency and the cosine similarity for treating missing user model term scores wins for the CMF.

**7.5.2 Baseline**

For comparing further runs a baseline is defined. This baseline uses the CMF learning from the Author Feature with a learning feature weight of 1. The baseline also scores messages the user is author of with 1.
7 Evaluation

<table>
<thead>
<tr>
<th>Configuration Option</th>
<th>Optimized Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Stemming</td>
<td>false</td>
</tr>
<tr>
<td>Use Word N-Grams</td>
<td>true</td>
</tr>
<tr>
<td>Use Char N-Grams</td>
<td>false</td>
</tr>
<tr>
<td>N-Grams Length</td>
<td>2</td>
</tr>
<tr>
<td>Do Tags</td>
<td>false</td>
</tr>
<tr>
<td>Do Tokens</td>
<td>true</td>
</tr>
<tr>
<td>Add Tags To Text</td>
<td>false</td>
</tr>
<tr>
<td>Term Weight Strategy</td>
<td>Inverse Message Frequency (imf)</td>
</tr>
<tr>
<td>Term Vector Similarity Strategy</td>
<td>Cosine Similarity</td>
</tr>
<tr>
<td>Treat Missing User Model Term</td>
<td>true</td>
</tr>
<tr>
<td>Scores As Zero</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.4: Optimized configuration for the information extraction and Content Match Feature (CMF).

7.6 Scoring Feature Optimization

For the feature-based approach without the CMF and CbMF only the scoring feature weights are used and must be optimized. For each feature weight $f_w$ values from 0 to 1 are possible. To use it with the GA a precision of 0.1 was used, hence 11 values are possible for each feature weight: 0, 0.1, 0.2, … 1. The scoring feature weights have been optimized using the GA on the optimization dataset. The optimized feature weights are shown in Table 7.5.

From these optimized feature weights the following can be observed. There are four features that show a high interest of the user in a message:

**Author Feature** This, in fact, is trivial; if the user is the author of the message, the message is relevant with a high probability for the author / user.

**Mention Feature** If the user is mentioned in the message, the message is relevant with a high probability.

**Discussion Mention Feature** If the user is mentioned within the discussion the message is part of, the message is relevant with a high probability.

**Discussion Participation Feature** If the user is mentioned within the discussion of the message, the message is probably relevant.

It is interesting to notice, that the Discussion Mention Feature reaches an even higher importance as the mention feature. Also, it is likely that a message is irrelevant for the user, if the message is not in a discussion the user is involved in and has no mentions for the user.

The evaluation results on the test dataset are shown and explained when discussing the content-based approach in the next Section 7.7.
7.7 Evaluation of Content-Based Approach

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature Weight $f_w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discussion Root Feature</td>
<td>0.3</td>
</tr>
<tr>
<td>Author Feature</td>
<td>0.9</td>
</tr>
<tr>
<td>Mention Feature</td>
<td>0.8</td>
</tr>
<tr>
<td>Discussion Participation Feature</td>
<td>0.6</td>
</tr>
<tr>
<td>Discussion Mention Feature</td>
<td>0.9</td>
</tr>
<tr>
<td>Discussion No Participation Feature</td>
<td>0.2</td>
</tr>
<tr>
<td>Discussion No Mention Feature</td>
<td>0.0</td>
</tr>
<tr>
<td>Content Match Feature</td>
<td>0.8</td>
</tr>
<tr>
<td>Collaboration Feature</td>
<td>0.9</td>
</tr>
</tbody>
</table>

The CMF and CbMF weights are shown for completeness here. How those values are optimized is described in Section 7.7.1 and Section 7.10.1 respectively.

Table 7.5: Optimized configuration for the scoring feature weights including CMF and CbMF.

7.7 Evaluation of Content-Based Approach

After the basic algorithm configuration is set up the content-based approach can be evaluated. The learning feature weights must be optimized first. An analysis of how a topic-based user model performs compared to a global user model is included.

7.7.1 Optimization

The optimization of the learning feature weights follows the same principle as the scoring feature weight optimization in Section 7.6. For each learning feature weight 11 values are possible, starting from 0 to 1 in increments of 0.1. With those possible values a GA has been applied on the optimizing dataset and the final optimized weights are shown in Table 7.6. The scores have been optimized against $F_1$ using RwoI. For Rall the scoring features have a higher influence and they make it more difficult to show the influence of the content-based approach.

The optimization already indicates that a topic-specific user model is the best since it was identified as a configuration of the optimum. The optimized score-to-learn threshold is 0.6 and therefore it is only learned if the feature weight is $\geq 0.6$. Therefore, terms of messages are integrated in the user model only when the following applies:

1. The user is author of the message (Author Feature).
2. The user is mentioned within the message (Mention Feature).
7 Evaluation

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature Weight ( \lambda_f w )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discussion Root Feature</td>
<td>0.1</td>
</tr>
<tr>
<td>Author Feature</td>
<td>0.7</td>
</tr>
<tr>
<td>Mention Feature</td>
<td>0.7</td>
</tr>
<tr>
<td>Discussion Participation Feature</td>
<td>0.8</td>
</tr>
<tr>
<td>Discussion Mention Feature</td>
<td>0.4</td>
</tr>
<tr>
<td>Discussion No Participation Feature</td>
<td>0.0</td>
</tr>
<tr>
<td>Discussion No Mention Feature</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Parameter | Value |
-----------|-------|
Score-to-Learn | 0.6   |
Minimum Clean Text Length | 0.5 |
Learn from all Parent Messages | true |
Use Topic-Specific User Model | true |
Learning Factor \( \alpha \) for Incremental User Model | 0.25 |

Table 7.6: Optimized configuration for learning a user model.

3. The user is author of a message within the discussion the message belongs to. (Discussion Participation Feature).

4. All messages that are recursive parent messages of one of the three types of messages above. If a message has the Author, Mention or Discussion Participation Feature and is a reply to another message, the parent message is also used for learning.

7.7.2 Results

The following different configuration runs are compared:

**Baseline** This is the baseline as defined in Section 7.5.2.

**Scoring Features** This configuration uses only the optimized scores as in Section 7.6 without the CMF or CbMF.

**Content-Based** This configuration uses the optimized scoring and learning weights as defined in Section 7.7.1.

**Content-Based Topic** This configuration uses a user model per topic per user.

**Content-Based Incremental** This configuration is based on the Content-Based configuration but learns the user model incremental (see Section 5.4.1).
7.7 Evaluation of Content-Based Approach

**Content-Based Incremental Topic** This configuration is based on the Content-Based Incremental configuration but uses a user model per topic.

In Table 7.7 the measures are shown for [Rall] in Table 7.8 for [RwoI] and in Table 7.10 for [RwoDI]. The measures used are TB-MAP and TB-P@k for the top 10 messages per day and per week and also the F1 and F2 scores. The highest value per measure is marked in bold.

The best values are reached using the content-based topic approach for nearly all measures and all rating sets. Only TB-P@k 10 Week of the baseline for [Rall] performs best. But the baseline itself performs very poorly for the other rating sets. The same applies for the scoring features approach. Also, the content-based approach without a topic-specific user model performs poorly. In Figure 7.9 the same values are shown for [RwoI] and in Figure 7.10 for [RwoDI]. From the tables and even more clearly in the figures it can be observed that the content-based topic approach outperforms all other runs. Only the non-content-based approach using only the scoring features comes close to reaching the performance for [Rall]. The content-based topic approach using the term count learning strategy performs slightly better than the incremental learning strategy.

<table>
<thead>
<tr>
<th>Run</th>
<th>F1 Score</th>
<th>F2 Score</th>
<th>TB-P@k 10 Day</th>
<th>TB-MAP 10 Day</th>
<th>TB-P@k 10 Week</th>
<th>TB-MAP 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.588</td>
<td>0.535</td>
<td>0.687</td>
<td>0.733</td>
<td>0.820</td>
<td>0.831</td>
</tr>
<tr>
<td>Scoring Features</td>
<td>0.619</td>
<td>0.598</td>
<td>0.723</td>
<td>0.748</td>
<td>0.783</td>
<td>0.837</td>
</tr>
<tr>
<td>Content-Based</td>
<td>0.582</td>
<td>0.584</td>
<td>0.698</td>
<td>0.724</td>
<td>0.786</td>
<td>0.833</td>
</tr>
<tr>
<td>Content-Based Topic</td>
<td>0.635</td>
<td>0.665</td>
<td>0.739</td>
<td>0.761</td>
<td>0.793</td>
<td>0.842</td>
</tr>
<tr>
<td>Content-Based Incremental</td>
<td>0.583</td>
<td>0.582</td>
<td>0.696</td>
<td>0.723</td>
<td>0.785</td>
<td>0.830</td>
</tr>
<tr>
<td>Content-Based Incremental Topic</td>
<td>0.629</td>
<td>0.650</td>
<td>0.735</td>
<td>0.756</td>
<td>0.793</td>
<td>0.842</td>
</tr>
</tbody>
</table>

Table 7.7: Results for content-based approaches for All Ratings (Rall).

<table>
<thead>
<tr>
<th>Run</th>
<th>F1 Score</th>
<th>F2 Score</th>
<th>TB-P@k 10 Day</th>
<th>TB-MAP 10 Day</th>
<th>TB-P@k 10 Week</th>
<th>TB-MAP 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.206</td>
<td>0.393</td>
<td>0.358</td>
<td>0.469</td>
<td>0.297</td>
<td>0.472</td>
</tr>
<tr>
<td>Scoring Features</td>
<td>0.265</td>
<td>0.401</td>
<td>0.460</td>
<td>0.584</td>
<td>0.533</td>
<td>0.654</td>
</tr>
<tr>
<td>Content-Based</td>
<td>0.251</td>
<td>0.412</td>
<td>0.433</td>
<td>0.554</td>
<td>0.527</td>
<td>0.639</td>
</tr>
<tr>
<td>Content-Based Topic</td>
<td>0.396</td>
<td>0.476</td>
<td>0.566</td>
<td>0.628</td>
<td>0.560</td>
<td>0.683</td>
</tr>
<tr>
<td>Content-Based Incremental</td>
<td>0.251</td>
<td>0.409</td>
<td>0.438</td>
<td>0.552</td>
<td>0.521</td>
<td>0.640</td>
</tr>
<tr>
<td>Content-Based Incremental Topic</td>
<td>0.380</td>
<td>0.473</td>
<td>0.548</td>
<td>0.616</td>
<td>0.560</td>
<td>0.686</td>
</tr>
</tbody>
</table>

Table 7.8: Results for content-based approaches for Ratings without Interaction (RwoI).
7 Evaluation

<table>
<thead>
<tr>
<th>Run</th>
<th>( F_1 ) Score</th>
<th>( F_2 ) Score</th>
<th>TB-P@k 10 Day</th>
<th>TB-MAP 10 Day</th>
<th>TB-P@k 10 Week</th>
<th>TB-MAP 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.180</td>
<td>0.354</td>
<td>0.374</td>
<td>0.465</td>
<td>0.272</td>
<td>0.498</td>
</tr>
<tr>
<td>Scoring Features</td>
<td>0.217</td>
<td>0.364</td>
<td>0.331</td>
<td>0.500</td>
<td>0.312</td>
<td>0.526</td>
</tr>
<tr>
<td>Content-Based</td>
<td>0.204</td>
<td>0.371</td>
<td>0.274</td>
<td>0.454</td>
<td>0.251</td>
<td>0.492</td>
</tr>
<tr>
<td>Content-Based Topic</td>
<td>0.332</td>
<td>0.430</td>
<td>0.531</td>
<td>0.605</td>
<td>0.483</td>
<td>0.617</td>
</tr>
<tr>
<td>Content-Based Incremental</td>
<td>0.201</td>
<td>0.371</td>
<td>0.283</td>
<td>0.451</td>
<td>0.248</td>
<td>0.495</td>
</tr>
<tr>
<td>Content-Based Incremental Topic</td>
<td>0.303</td>
<td>0.426</td>
<td>0.503</td>
<td>0.587</td>
<td>0.446</td>
<td>0.594</td>
</tr>
</tbody>
</table>

Table 7.9: Results for content-based approaches for Ratings without Discussion Interaction (RwoDI).

Figure 7.9: Evaluation results for feature- and content-based runs for Ratings without Interaction (RwoI).

7.7.3 Interpretation

The topic-specific user model clearly outperforms all other runs. Using the same optimized configuration without a global user model shows very poor results. The advantage of the content-based topic approach gets bigger as...
fewer scoring features can be applied in the different rating sets. The scoring features have a huge impact on RwI where most of the ratings are relevant. Because RwI is included in Rall, Rall also leads to very good results for this approach without using the CMF or CbMF. However, as those features are no longer available, the gap between the scoring features and the content-based topic approach widens. This also shows that it is much harder to identify relevant messages for rating sets RwoI and RwoDI. However, for TB-P@53, 1% of the top 10 messages per day are retrieved as relevant in the content-based topic approach. The lower $F_1$ and $F_2$ values indicate that the approach helps to identify the top elements, while there are, however, some messages that are hard or even impossible to predict. These are probably messages that are not relevant for a user based on the content, but on other information which is unknown to the algorithm.

The incremental user model gives slightly lower results than the term count user model learning strategy. Since the incremental user model learning strategy has more impact on the youngest messages integrated into the user
model, it indicates that it is better to maintain all messages observed for a longer period of time.

Comparing all measures, the score for $TB-P@k_{10}$-Week for $Rall$ of the baseline run is an exception because all other measure of this run are worse than the content-based topic-specific approach. The content-based topic approach is the best, outperforming all other runs. Therefore, this approach is used for the comparison for all of the next runs.

7.8 Evaluation of User Model Adaptation Approach

In this section the User Model Adaptation (UMA) approach is analyzed and compared to the content-based runs. As before, first the necessary parameters for the UMA approach are optimized and secondly, the evaluation runs are executed on the test dataset.

7.8.1 Optimization

The best parameters have been determined independently for the three variants: user-model-based similarity (UMS), topic similarity (TS) and combined similarity (CS). The optimization is based on the best content-based run using a topic-specific user model. Hence, all runs in this section use a topic-specific user model. As before, the optimization was run on the 30% of the dataset using $RwoI$. The optimized parameter values for the three variants are shown in Table 7.10.

It is interesting to note that it is better to take only the top 3 users or topics for similarity than to take all and average then. Only the 3 most similar users or topics are used to determine the terms for adaptation.

7.8.2 Results

For each of the different UMA variants, the best top users identified by the optimization and for all users are used. The results for $Rall$ are shown in Table 7.11, for $RwoI$ in 7.12 and for $RwoDI$ in 7.13. In Figure 7.11 the same results are shown for $RwoI$.

For nearly all measures and rating sets the UMA variant using the user model similarity performs best. Only for $TB-P@k_{10}$-Week of the baseline run a non-UMA approach is the best. Also, for $TB-P@k_{10}$-Week the combined similarity (using both topic and user models to adapt) gives the best result. For $TB-MAP_{10}$-Week of $RwoDI$ the content-based topic approach is best, but for $TB-P@k_{10}$-Week it is the UMA variant using the user model similarity.

The UMA using the user model similarity with the top 3 users gives the best steady result for all measures. It is either the best value or close to the
### 7.8 Evaluation of User Model Adaptation Approach

<table>
<thead>
<tr>
<th>Parameter for UMA</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>General UMA</td>
<td></td>
</tr>
<tr>
<td>Confidence Threshold $\zeta$</td>
<td>0.6</td>
</tr>
<tr>
<td>Score Threshold $\rho$</td>
<td>0.9</td>
</tr>
<tr>
<td>User-Similarity-Based UMA</td>
<td></td>
</tr>
<tr>
<td>Interest Term Threshold $\iota$</td>
<td>0.5</td>
</tr>
<tr>
<td>Top-$n$ similar Users</td>
<td>3</td>
</tr>
<tr>
<td>Topic-Similarity-Based UMA</td>
<td></td>
</tr>
<tr>
<td>Top-$n$ similar Topics</td>
<td>3</td>
</tr>
<tr>
<td>Combined User-Topic-Similarity UMA</td>
<td></td>
</tr>
<tr>
<td>Maximum Votes $\gamma$</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 7.10: Optimized configuration for UMA.

best value, except for $TB-P_{@k}$-10-Week of Rall (Baseline) and $TB-MAP$-10-Week for Kwol (UMA UMS All) and Kwol (Content-Based Topic). Therefore this variant is also better than the content-based topic approach.

The UMA variant with topic similarity does not reach any top result for any measure. It does not achieve the best results. Using all topics lead to worse results than the content-based topic approach. Using the top 3 topics leads to similar results as the content-based topic approach.

The results of the combined similarity of the UMA approach are in between the user model and topic similarity variants. Using all users and topics performs poorly, even worse than if the UMA in the content-based topic approach was not used. Using the top 3 users and topics at the same time gives slightly better results than the content-based topic approach, but not as good as using only the top 3 users.

**Top-$n$ Users and Topics**

The UMA approach using the top 3 most similar users gives the best result. In Table 7.14 and 7.15 this variant is compared with different top user configurations for Rall and Kwol, respectively.

The results for different users are rather similar. Only the top 1 and all users give different results. Using only one user for adaptation leads to lower measures. It is different for all users. For some measures (e.g. $F_2$, $TB-P_{@k}$-10-Week, $TB-MAP$-10-Week of Kwol) it gives the best results,
7 Evaluation

<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>TB-P@k 10 Day</th>
<th>TB-MAP 10 Day</th>
<th>TB-P@k 10 Week</th>
<th>TB-MAP 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.588</td>
<td>0.535</td>
<td>0.687</td>
<td>0.733</td>
<td><strong>0.820</strong></td>
<td>0.831</td>
</tr>
<tr>
<td>CB$^a$ Topic</td>
<td>0.635</td>
<td>0.665</td>
<td>0.739</td>
<td>0.761</td>
<td>0.793</td>
<td>0.842</td>
</tr>
<tr>
<td>UMA$^b$, UMS$^c$, All</td>
<td>0.646</td>
<td>0.684</td>
<td>0.739</td>
<td>0.768</td>
<td>0.795</td>
<td><strong>0.847</strong></td>
</tr>
<tr>
<td>UMA$^b$, UMS$^c$, Top 3</td>
<td><strong>0.650</strong></td>
<td><strong>0.692</strong></td>
<td><strong>0.774</strong></td>
<td><strong>0.775</strong></td>
<td>0.765</td>
<td>0.817</td>
</tr>
<tr>
<td>UMA$^b$, TS$^d$, All</td>
<td>0.581</td>
<td>0.558</td>
<td>0.696</td>
<td>0.721</td>
<td>0.785</td>
<td>0.831</td>
</tr>
<tr>
<td>UMA$^b$, TS$^d$, Top 3</td>
<td>0.631</td>
<td>0.663</td>
<td>0.736</td>
<td>0.760</td>
<td>0.793</td>
<td>0.842</td>
</tr>
<tr>
<td>UMA$^b$, CS$^e$, All</td>
<td>0.580</td>
<td>0.586</td>
<td>0.701</td>
<td>0.730</td>
<td>0.785</td>
<td>0.838</td>
</tr>
<tr>
<td>UMA$^b$, CS$^e$, Top 3</td>
<td>0.646</td>
<td>0.681</td>
<td>0.738</td>
<td>0.767</td>
<td>0.795</td>
<td>0.845</td>
</tr>
</tbody>
</table>

$^a$ Content-Based; $^b$ User Model Adaptation; $^c$ User-Model-Based Similarity; $^d$ Topic Similarity; $^e$ Combined Similarity using Topics and Users

Table 7.11: Results for UMA approaches for All Ratings (Rall).

but again for others it gives the lowest in this comparison (TB-P@k 10-Day, TB-MAP 10-Day of RwoDI). The best stable results are achieved using the top 3 and 4 users. However, the difference in using more top users is small.

7.8.3 Interpretation

On the one hand, the results clearly show that using the UMA approach with user-model-based similarity improves the quality of recommendation. On the other hand, the topic similarity does not improve the quality. The interpretation of the content-based approach shows that a topic-specific model works best. That means that the interest of terms is treated differently for each topic and hence the interests are different for each user per topic. Therefore, for adapting the user model, it is better to use the same topic instead of trying other topics to adapt. The user-model-based similarity only consults the same term of the same topic but in different user models.

The combined similarity takes its strength from the user-model-based similarity. The fact that it does not reach the same quality indicates that even in this combined case, the adaptation using topics does not improve the quality of the recommendation. Using only the user similarity variant is sufficient to achieve the improvement shown.

Using all users for adaptation does not work as well as using only the most similar users. If all users are used, the adaptation is greedy, trying to adapt as much as possible. This leads to a high $F_2$ for RwoDI but a low $F_1$. If only the most similar users are used the adaptation will be more selective, avoiding to recommending too many irrelevant messages.
### 7.8 Evaluation of User Model Adaptation Approach

#### Table 7.12: Results for UMA approaches for Ratings without Interaction (RwoI).

<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>$TB@P@k$ 10 Day</th>
<th>$TB-MAP$ 10 Day</th>
<th>$TB@P@k$ 10 Week</th>
<th>$TB-MAP$ 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.206</td>
<td>0.393</td>
<td>0.358</td>
<td>0.469</td>
<td>0.297</td>
<td>0.472</td>
</tr>
<tr>
<td>CB\textsuperscript{a}</td>
<td>0.396</td>
<td>0.476</td>
<td>0.566</td>
<td>0.628</td>
<td>0.560</td>
<td>0.683</td>
</tr>
<tr>
<td>UMA\textsuperscript{b}, UMS\textsuperscript{c}, All</td>
<td>0.414</td>
<td>0.486</td>
<td>0.563</td>
<td>0.634</td>
<td>0.569</td>
<td>0.697</td>
</tr>
<tr>
<td>UMA\textsuperscript{b}, UMS\textsuperscript{c}, Top 3</td>
<td>0.432</td>
<td>0.479</td>
<td>\textit{0.650}</td>
<td>\textit{0.668}</td>
<td>0.559</td>
<td>0.655</td>
</tr>
<tr>
<td>UMA\textsuperscript{b}, Ts\textsuperscript{d}, All</td>
<td>0.240</td>
<td>0.407</td>
<td>0.429</td>
<td>0.546</td>
<td>0.519</td>
<td>0.637</td>
</tr>
<tr>
<td>UMA\textsuperscript{b}, Ts\textsuperscript{d}, Top 3</td>
<td>0.389</td>
<td>0.475</td>
<td>0.558</td>
<td>0.624</td>
<td>0.563</td>
<td>0.684</td>
</tr>
<tr>
<td>UMA\textsuperscript{b}, CS\textsuperscript{e}, All</td>
<td>0.261</td>
<td>0.408</td>
<td>0.445</td>
<td>0.560</td>
<td>0.527</td>
<td>0.656</td>
</tr>
<tr>
<td>UMA\textsuperscript{b}, CS\textsuperscript{e}, Top 3</td>
<td>0.414</td>
<td>0.486</td>
<td>0.571</td>
<td>0.637</td>
<td>\textbf{0.571}</td>
<td>0.691</td>
</tr>
</tbody>
</table>

\textsuperscript{a} Content-Based; \textsuperscript{b} User Model Adaptation; \textsuperscript{c} User-Model-Based Similarity; \textsuperscript{d} Topic Similarity; \textsuperscript{e} Combined Similarity using Topics and Users

#### Table 7.13: Results for UMA approaches for Ratings without Discussion Interaction (RwoDI).

<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>$TB@P@k$ 10 Day</th>
<th>$TB-MAP$ 10 Day</th>
<th>$TB@P@k$ 10 Week</th>
<th>$TB-MAP$ 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.180</td>
<td>0.354</td>
<td>0.374</td>
<td>0.465</td>
<td>0.272</td>
<td>0.498</td>
</tr>
<tr>
<td>CB\textsuperscript{a}</td>
<td>0.332</td>
<td>0.430</td>
<td>0.531</td>
<td>0.605</td>
<td>0.483</td>
<td>\textbf{0.617}</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, Top 1</td>
<td>0.360</td>
<td>0.443</td>
<td>0.555</td>
<td>0.615</td>
<td>0.462</td>
<td>0.603</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, Top 3</td>
<td>0.377</td>
<td>0.432</td>
<td>\textbf{0.612}</td>
<td>\textbf{0.649}</td>
<td>\textbf{0.509}</td>
<td>0.601</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, Ts\textsuperscript{d}, All</td>
<td>0.193</td>
<td>0.369</td>
<td>0.264</td>
<td>0.444</td>
<td>0.248</td>
<td>0.485</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, Ts\textsuperscript{d}, Top 3</td>
<td>0.326</td>
<td>0.429</td>
<td>0.521</td>
<td>0.598</td>
<td>0.473</td>
<td>0.614</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, CS\textsuperscript{e}, All</td>
<td>0.196</td>
<td>0.370</td>
<td>0.294</td>
<td>0.461</td>
<td>0.257</td>
<td>0.503</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, CS\textsuperscript{e}, Top 3</td>
<td>0.357</td>
<td>0.442</td>
<td>0.560</td>
<td>0.618</td>
<td>0.458</td>
<td>0.597</td>
</tr>
</tbody>
</table>

\textsuperscript{a} Content-Based; \textsuperscript{b} User Model Adaptation; \textsuperscript{c} User-Model-Based Similarity; \textsuperscript{d} Topic Similarity; \textsuperscript{e} Combined Similarity using Topics and Users

#### Table 7.14: Results for UMA approaches using User-Model-Based Similarity (UMS) with different top-n user configurations for All Ratings (Rall).

<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>$TB@P@k$ 10 Day</th>
<th>$TB-MAP$ 10 Day</th>
<th>$TB@P@k$ 10 Week</th>
<th>$TB-MAP$ 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, Top 1</td>
<td>0.640</td>
<td>0.673</td>
<td>0.770</td>
<td>0.773</td>
<td>0.761</td>
<td>0.810</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, Top 2</td>
<td>0.650</td>
<td>0.687</td>
<td>\textbf{0.778}</td>
<td>\textbf{0.776}</td>
<td>0.765</td>
<td>0.817</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, Top 3</td>
<td>0.650</td>
<td>\textbf{0.692}</td>
<td>0.774</td>
<td>0.775</td>
<td>0.765</td>
<td>0.817</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, Top 4</td>
<td>0.651</td>
<td>0.691</td>
<td>0.774</td>
<td>0.775</td>
<td>0.767</td>
<td>0.818</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, Top 5</td>
<td>0.652</td>
<td>0.691</td>
<td>0.774</td>
<td>0.775</td>
<td>0.767</td>
<td>0.818</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, Top 7</td>
<td>0.653</td>
<td>0.691</td>
<td>0.775</td>
<td>0.775</td>
<td>0.767</td>
<td>0.818</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, Top 10</td>
<td>0.653</td>
<td>0.692</td>
<td>0.775</td>
<td>0.775</td>
<td>0.767</td>
<td>0.818</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, Top 15</td>
<td>0.653</td>
<td>0.691</td>
<td>0.774</td>
<td>0.775</td>
<td>0.767</td>
<td>0.818</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, Top 20</td>
<td>0.652</td>
<td>0.691</td>
<td>0.774</td>
<td>0.775</td>
<td>0.767</td>
<td>0.818</td>
</tr>
<tr>
<td>UMA\textsuperscript{a}, UMS\textsuperscript{b}, All Users</td>
<td>0.646</td>
<td>0.684</td>
<td>0.739</td>
<td>0.768</td>
<td>\textbf{0.795}</td>
<td>\textbf{0.847}</td>
</tr>
</tbody>
</table>

\textsuperscript{a} User Model Adaptation; \textsuperscript{b} User-Model-Based Similarity
Figure 7.11: Evaluation results for UMA variants for RwoI.

* Content-Based;
* User Model Adaptation;
* User-Model-Based Similarity;
* Topic Similarity;
* Combined Similarity using Topics and Users;
7.9 Evaluation of Short-Term and Long-Term Approach

In this section the short-term approach is evaluated. As before, the relevant parameters are first optimized and then evaluated on the test dataset. Finally, the results are interpreted.

7.9.1 Optimization

The short-term approach is based on the best content-based approach using a topic-specific user model. Hence, for all runs in this section a topic-specific user model was used. For the decay variant fixed parameters were used, one for a day and one for a week. No other parameters are optimized in this case. For the time-binned user model bins of a day and a week were used. For the day bin user model 30 bins were used and 4 bins for a week, to closely match a month.

For the detection of long-terms and the aggregation of the short- and long-term user models the optimization has been run on 30% of the dataset. The optimized configuration and for completeness, the fixed configuration are shown in Table 7.16.

The optimized long-term detection identifies periodic and permanent terms. The permanent detector actually takes all short user model terms with $um_u(t) \geq 0.8$ and transfers them to the long-term user model. The periodic detector considers user model terms with $um_u(t) \geq 0.5$ where the term occurs in at least two bins with at least one bin distance. The user models are first merged per term. The merged scores are then used to compute the CMI.
7 Evaluation

<table>
<thead>
<tr>
<th>Parameter for Short-Term</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Day Decay</strong></td>
<td></td>
</tr>
<tr>
<td>Decay Factor $\kappa$</td>
<td>1 day</td>
</tr>
<tr>
<td><strong>Week Decay</strong></td>
<td></td>
</tr>
<tr>
<td>Decay Factor $\kappa$</td>
<td>7 days</td>
</tr>
<tr>
<td><strong>Time-Binned User Model Day</strong></td>
<td></td>
</tr>
<tr>
<td>Bin Length</td>
<td>1 Day</td>
</tr>
<tr>
<td>Number of Bins</td>
<td>30</td>
</tr>
<tr>
<td><strong>Time-Binned User Model Week</strong></td>
<td></td>
</tr>
<tr>
<td>Bin Length</td>
<td>7 Days</td>
</tr>
<tr>
<td>Number of Bins</td>
<td>4</td>
</tr>
<tr>
<td><strong>Short-, Long-Term User Models</strong></td>
<td></td>
</tr>
<tr>
<td>Bin Length</td>
<td>7 Days</td>
</tr>
<tr>
<td>Number of Bins</td>
<td>4</td>
</tr>
<tr>
<td>Periodic Occurrence Count</td>
<td>2</td>
</tr>
<tr>
<td>Periodic Occurrence Distance</td>
<td>1</td>
</tr>
<tr>
<td>Periodic Interest Term Threshold</td>
<td>0.5</td>
</tr>
<tr>
<td>Permanent Occurrence Minimum Length</td>
<td>1</td>
</tr>
<tr>
<td>Permanent Interest Term Threshold</td>
<td>0.8</td>
</tr>
<tr>
<td>User Model Merging Strategy</td>
<td>$CMF_{TermMerge^{Weighted}}$</td>
</tr>
<tr>
<td>Short-Term User Model Weight $\omega_{ST}$</td>
<td>0.5</td>
</tr>
</tbody>
</table>

See Equations 5.34, 5.32.

Table 7.16: Optimized configuration for short-term approach.

### 7.9.2 Results

All runs are executed on the test dataset. The different short-term approaches are compared in Table 7.17 for $R_{all}$, in Table 7.18 for $R_{woI}$ and in Table 7.19 for $R_{woDI}$. In all comparisons the baseline and the best content-based approach (using a topic-specific user model) are used. Also, two runs using a decaying user model and two runs using a time-binned user model are used. Using all four of these runs does not lead to an improvement for all rating sets. The user model with decay decreases the performance compared to the content-based topic approach. The week decay is better than the
7.9 Evaluation of Short-Term and Long-Term Approach

### Table 7.17: Results for short-term approaches for All Ratings *(Rall)*.

<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>$TB$-$P@k$ 10 Day</th>
<th>$TB$-$MAP$ 10 Day</th>
<th>$TB$-$P@k$ 10 Week</th>
<th>$TB$-$MAP$ 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.588</td>
<td>0.535</td>
<td>0.687</td>
<td>0.733</td>
<td>0.820</td>
<td>0.831</td>
</tr>
<tr>
<td>CB$^a$ Topic</td>
<td>0.635</td>
<td>0.665</td>
<td>0.739</td>
<td>0.761</td>
<td>0.793</td>
<td>0.842</td>
</tr>
<tr>
<td>CB$^a$ Topic, Day Decay</td>
<td>0.620</td>
<td>0.630</td>
<td>0.733</td>
<td>0.754</td>
<td>0.790</td>
<td>0.840</td>
</tr>
<tr>
<td>CB$^a$ Topic, Week Decay</td>
<td>0.629</td>
<td>0.651</td>
<td>0.736</td>
<td>0.756</td>
<td>0.791</td>
<td>0.841</td>
</tr>
<tr>
<td>TB$^b$ UM$^c$ Day</td>
<td>0.619</td>
<td>0.602</td>
<td>0.723</td>
<td>0.749</td>
<td>0.786</td>
<td>0.838</td>
</tr>
<tr>
<td>TB$^b$ UM$^c$ Week</td>
<td>0.624</td>
<td>0.610</td>
<td>0.725</td>
<td>0.752</td>
<td>0.786</td>
<td>0.838</td>
</tr>
<tr>
<td>ST$^d$ &amp; LT$^e$ UM$^c$</td>
<td><strong>0.635</strong></td>
<td><strong>0.665</strong></td>
<td>0.739</td>
<td><strong>0.761</strong></td>
<td>0.793</td>
<td><strong>0.842</strong></td>
</tr>
</tbody>
</table>

*a* Content-Based;  
*b* Time-Binned;  
*c* User Model;  
*d* Short-Term;  
*e* Long-Term

### Table 7.18: Results for short-term approaches for Ratings without Interaction *(RwoI)*.

<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>$TB$-$P@k$ 10 Day</th>
<th>$TB$-$MAP$ 10 Day</th>
<th>$TB$-$P@k$ 10 Week</th>
<th>$TB$-$MAP$ 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.206</td>
<td>0.393</td>
<td>0.358</td>
<td>0.469</td>
<td>0.297</td>
<td>0.472</td>
</tr>
<tr>
<td>CB$^a$ Topic</td>
<td>0.396</td>
<td>0.476</td>
<td><strong>0.566</strong></td>
<td><strong>0.628</strong></td>
<td><strong>0.560</strong></td>
<td><strong>0.683</strong></td>
</tr>
<tr>
<td>CB$^a$ Topic, Day Decay</td>
<td>0.343</td>
<td>0.462</td>
<td>0.526</td>
<td>0.612</td>
<td>0.548</td>
<td>0.662</td>
</tr>
<tr>
<td>CB$^a$ Topic, Week Decay</td>
<td>0.381</td>
<td>0.472</td>
<td>0.550</td>
<td>0.615</td>
<td>0.554</td>
<td>0.674</td>
</tr>
<tr>
<td>TB$^b$ UM$^c$ Day</td>
<td>0.268</td>
<td>0.405</td>
<td>0.466</td>
<td>0.589</td>
<td>0.533</td>
<td>0.659</td>
</tr>
<tr>
<td>TB$^b$ UM$^c$ Week</td>
<td>0.278</td>
<td>0.420</td>
<td>0.483</td>
<td>0.600</td>
<td>0.529</td>
<td>0.653</td>
</tr>
<tr>
<td>ST$^d$ &amp; LT$^e$ UM$^c$</td>
<td><strong>0.396</strong></td>
<td><strong>0.476</strong></td>
<td>0.566</td>
<td>0.628</td>
<td>0.560</td>
<td>0.683</td>
</tr>
</tbody>
</table>

*a* Content-Based;  
*b* Time-Binned;  
*c* User Model;  
*d* Short-Term;  
*e* Long-Term

Using short- and long-term user models leads to similar results as the content-based topic approach. The difference is only in the magnitude of a thousandth, i.e. barely noticeable. The optimized long-term detection is not very restrictive in a form where many of the terms are transferred from short-term to long-term user model.

### 7.9.3 Interpretation

None of the approaches leads to a noticeable improvement in the quality for the result. Using a decay of the user model does not improve the results. The week decaying user model is better than the day decaying one, indicating that using no decay is the best. This is the case in the content-based topic approach. That the time-binned user models do not perform better is not surprising since in that case only terms within a month are kept. However, the time-binned user models are used as short-term user models. The aston-
lishing fact is that even the more sophisticated approach of separating the short and long-term interest did not lead to an improvement. There could be several reasons for this:

- The detection and separation of long-term interest is not sensitive enough.
- The underlying data does not allow the identification and separation of short- and long-term interest.
- There are no short or long-term interests that help to improve the quality.

The approach used to identify short- and long-term interest is based on reoccurring interest terms in different time bins. The approach can be extended to consider other detection patterns for short and long-term interests.

The second reason suggests that using only the feature to learn and maintain a user model and the interaction of the user with the system is not enough to detect short-term interest at all. In that case other data sources, e.g. direct user feedback, must be used to achieve better results.

Furthermore, the third reason proposes that there is an advantage in detecting and handling short- and long-term interest differently. This contradicts the results of the pre-analysis in Appendix A. It is likely that the true reason is somewhere in between the first and second. This interpretation needs further research.

### 7.10 Evaluation of Collaboration-Based Approach

In this section the collaborative-based approach is compared to the best content-based one. The collaborative-based recommender learns using the same feature weights as the content-based approach.
7.10 Evaluation of Collaboration-Based Approach

### 7.10.1 Optimization

The only optimization parameter for the collaborative-based approach is the scoring weight of the Collaboration Match Feature (CbMF). The weight has been optimized on 30% of the dataset for RwoI. The optimized weight has already been shown in Table 7.5.

### 7.10.2 Results

All runs used the test dataset. The results are shown in Table 7.20 for Rall in Table 7.21 for RwoI and in Table 7.22 for RwoDI. In all three rating sets only the UtM topic-specific variant of the CbMF gives competitive results. This variant leads to the best $F_2$ for all rating sets in this comparison but does not reach the measure values for any other measures compared to the content-based topic approach.

For the UtM variants the topic-specific variant is clearly better than the other. For the UtT variants there is only a slight difference between using the topic-specific terms and not using them.

### 7.10.3 Interpretation

The UtM collaborative-based recommender learns connections between users and messages based on the learned features. Therefore, the recommender only gives acceptable results on the messages that are part of the features used for learning. This is already partly covered by the scoring features. The results for Rall are mainly based on the scoring features, and the CbMF does not add much value to it.

---

Table 7.20: Results for collaboration-based approaches for All Ratings (Rall).

<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>TB-P@k 10 Day</th>
<th>TB-MAP 10 Day</th>
<th>TB-P@k 10 Week</th>
<th>TB-MAP 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.588</td>
<td>0.535</td>
<td>0.687</td>
<td>0.733</td>
<td>0.820</td>
<td>0.831</td>
</tr>
<tr>
<td>CB$^a$ Topic</td>
<td>0.635</td>
<td>0.665</td>
<td>0.739</td>
<td>0.761</td>
<td>0.793</td>
<td>0.842</td>
</tr>
<tr>
<td>CbB$^b$ UtM$^c$, Slope One</td>
<td>0.619</td>
<td>0.598</td>
<td>0.723</td>
<td>0.748</td>
<td>0.783</td>
<td>0.837</td>
</tr>
<tr>
<td>CbB$^b$ UtM$^c$ Topic, Slope One</td>
<td>0.619</td>
<td>0.683</td>
<td>0.736</td>
<td>0.748</td>
<td>0.788</td>
<td>0.839</td>
</tr>
<tr>
<td>CbB$^b$ UtT$^d$, Slope One</td>
<td>0.324</td>
<td>0.544</td>
<td>0.317</td>
<td>0.498</td>
<td>0.356</td>
<td>0.536</td>
</tr>
<tr>
<td>CbB$^b$ UtT$^d$ Topic, Slope One</td>
<td>0.341</td>
<td>0.550</td>
<td>0.332</td>
<td>0.514</td>
<td>0.363</td>
<td>0.538</td>
</tr>
</tbody>
</table>

$^a$ Content-Based; $^b$ Collaboration-Based; $^c$ User to Message; $^d$ User To Term;

As mentioned in Section 6.2.2 the collaboration algorithm runs once per day since a non-incremental implementation is used. To reflect the evaluation correctly, the test ratings are evaluated after the daily recomputation and not directly after the message occurred.
7 Evaluation

<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>TB-P@k 10 Day</th>
<th>TB-MAP 10 Day</th>
<th>TB-P@k 10 Week</th>
<th>TB-MAP 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.206</td>
<td>0.393</td>
<td>0.358</td>
<td>0.469</td>
<td>0.297</td>
<td>0.472</td>
</tr>
<tr>
<td>CB\textsuperscript{a} Topic</td>
<td>\textbf{0.396}</td>
<td>0.476</td>
<td>0.566</td>
<td>0.628</td>
<td>0.560</td>
<td>0.683</td>
</tr>
<tr>
<td>CB\textsuperscript{b} Ut\textsuperscript{c}\textsuperscript{m} \textsuperscript{d}, Slope One</td>
<td>0.256</td>
<td>0.390</td>
<td>0.452</td>
<td>0.578</td>
<td>0.524</td>
<td>0.647</td>
</tr>
<tr>
<td>CB\textsuperscript{b} Ut\textsuperscript{c}\textsuperscript{m} \textsuperscript{d} Topic, Slope One</td>
<td>0.328</td>
<td>\textbf{0.530}</td>
<td>0.500</td>
<td>0.588</td>
<td>0.550</td>
<td>0.672</td>
</tr>
<tr>
<td>CB\textsuperscript{b} Ut\textsuperscript{d}\textsuperscript{d}, Slope One</td>
<td>0.206</td>
<td>0.392</td>
<td>0.235</td>
<td>0.444</td>
<td>0.226</td>
<td>0.464</td>
</tr>
<tr>
<td>CB\textsuperscript{b} Ut\textsuperscript{d}\textsuperscript{d} Topic, Slope One</td>
<td>0.210</td>
<td>0.395</td>
<td>0.245</td>
<td>0.460</td>
<td>0.236</td>
<td>0.467</td>
</tr>
</tbody>
</table>

\textsuperscript{a} Content-Based; \textsuperscript{b} Collaboration-Based; \textsuperscript{c} User to Message; \textsuperscript{d} User To Term

Table 7.21: Results for collaboration-based approaches for Ratings without Interaction (RwoI).

<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>TB-P@k 10 Day</th>
<th>TB-MAP 10 Day</th>
<th>TB-P@k 10 Week</th>
<th>TB-MAP 10 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.180</td>
<td>0.354</td>
<td>0.374</td>
<td>0.465</td>
<td>0.272</td>
<td>0.498</td>
</tr>
<tr>
<td>CB\textsuperscript{b} Topic</td>
<td>\textbf{0.332}</td>
<td>0.430</td>
<td>0.531</td>
<td>0.605</td>
<td>\textbf{0.483}</td>
<td>\textbf{0.617}</td>
</tr>
<tr>
<td>CB\textsuperscript{b} Ut\textsuperscript{c}\textsuperscript{m} \textsuperscript{d}, Slope One</td>
<td>0.210</td>
<td>0.353</td>
<td>0.323</td>
<td>0.493</td>
<td>0.299</td>
<td>0.518</td>
</tr>
<tr>
<td>CB\textsuperscript{b} Ut\textsuperscript{c}\textsuperscript{m} \textsuperscript{d} Topic, Slope One</td>
<td>0.298</td>
<td>\textbf{0.494}</td>
<td>0.398</td>
<td>0.512</td>
<td>0.359</td>
<td>0.561</td>
</tr>
<tr>
<td>CB\textsuperscript{b} Ut\textsuperscript{d}\textsuperscript{d}, Slope One</td>
<td>0.181</td>
<td>0.355</td>
<td>0.217</td>
<td>0.428</td>
<td>0.185</td>
<td>0.463</td>
</tr>
<tr>
<td>CB\textsuperscript{b} Ut\textsuperscript{d}\textsuperscript{d} Topic, Slope One</td>
<td>0.185</td>
<td>0.357</td>
<td>0.231</td>
<td>0.444</td>
<td>0.211</td>
<td>0.476</td>
</tr>
</tbody>
</table>

\textsuperscript{a} Content-Based; \textsuperscript{b} Collaboration-Based; \textsuperscript{c} User to Message; \textsuperscript{d} User To Term

Table 7.22: Results for collaboration-based approaches for Ratings without Discussion Interaction (RwoDI).
It is interesting to notice that the UtM topic-specific variants give the best results for the collaborative-based approach. Hence, the differentiation into topics also works here, showing that the interest in messages and the user-to-message structure are different for different topics. This is not the case for the UtT variants. But here the low scores of UtT in all three rating sets show that this variant does not work at all. The idea of using the collaboration on user-to-term connections does not help to find relevant messages. Comparing all measures, the scores for $F_2$ of the UtM collaborative-based topic-specific run are an exception because all other measure of this run are worse than the content-based topic-specific approach.

7.11 Comparison of Different Approaches

In this section the best runs of the different approaches are compared to each other. The runs are compared in Table 7.23 and Figure 7.12 for Rall, in Table 7.24 and Figure 7.13 for RwoI, and in Table 7.25 and Figure 7.14 for RwoDI.

The runs compared are the baseline, the non-content-based approach using only the scoring feature weights, the content-based topic approach, the user model adaptation approach using a user-model-based similarity, the short- and long-term user model approach and, finally, the UtM collaborative-based topic-specific approach.

In the sections before only two top time-binned measures (10-Day and 10-Week) have been used for comparison. At this point, time-binned measures for 5-Day, 10-Day, 10-Week and 20-Week are used for comparison.

The biggest improvement is using a topic-specific user model for the content-based approach. Furthermore, the user model adaptation using the user model similarity adds a quality improvement on top of that. The short- and long-term user models do not bring an improvement compared to the content-based approach. The collaborative approach is not able to keep up with the best content-based approaches at all.

The topic-based and user model adaptation approach show its strength as less features are available for scoring. For both approaches, the difference to the scoring feature results is larger for RwoDI compared to Rall.
<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>TB-P@k 5 Day</th>
<th>TB-MAP 5 Day</th>
<th>TB-P@k 10 Day</th>
<th>TB-MAP 10 Day</th>
<th>TB-P@k 10 Week</th>
<th>TB-MAP 10 Week</th>
<th>TB-P@k 20 Week</th>
<th>TB-MAP 20 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.588</td>
<td>0.535</td>
<td>0.720</td>
<td>0.747</td>
<td>0.687</td>
<td>0.733</td>
<td><strong>0.820</strong></td>
<td>0.831</td>
<td><strong>0.796</strong></td>
<td>0.843</td>
</tr>
<tr>
<td>Scoring Features</td>
<td>0.619</td>
<td>0.598</td>
<td>0.709</td>
<td>0.710</td>
<td>0.723</td>
<td>0.748</td>
<td>0.783</td>
<td>0.837</td>
<td>0.757</td>
<td>0.806</td>
</tr>
<tr>
<td>CB$^a$ Topic</td>
<td>0.635</td>
<td>0.665</td>
<td>0.728</td>
<td>0.728</td>
<td>0.739</td>
<td>0.761</td>
<td>0.793</td>
<td><strong>0.842</strong></td>
<td>0.766</td>
<td>0.816</td>
</tr>
<tr>
<td>UMA$^a$, UMS$^c$, Top 3</td>
<td><strong>0.650</strong></td>
<td>0.692</td>
<td><strong>0.746</strong></td>
<td>0.732</td>
<td><strong>0.774</strong></td>
<td><strong>0.775</strong></td>
<td>0.765</td>
<td>0.817</td>
<td>0.745</td>
<td>0.814</td>
</tr>
<tr>
<td>ST$^d$ &amp; LT$^e$ UM$^f$</td>
<td>0.635</td>
<td>0.665</td>
<td>0.728</td>
<td>0.728</td>
<td>0.739</td>
<td>0.761</td>
<td>0.793</td>
<td>0.842</td>
<td>0.767</td>
<td>0.816</td>
</tr>
<tr>
<td>Chb$^g$, UtM$^h$, Topic, Slope One</td>
<td>0.619</td>
<td>0.683</td>
<td>0.720</td>
<td>0.719</td>
<td>0.736</td>
<td>0.748</td>
<td>0.788</td>
<td>0.839</td>
<td>0.765</td>
<td>0.811</td>
</tr>
</tbody>
</table>

$^a$ Content-Based; $^b$ User Model Adaptation; $^c$ User-Model-Based Similarity; $^d$ Short-Term; $^e$ Long-Term; $^f$ User Model; $^g$ Collaboration-Based; $^h$ User to Message

Table 7.23: Results for various top time-binned measures for best of different approaches for All Ratings (Rall).

<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>TB-P@k 5 Day</th>
<th>TB-MAP 5 Day</th>
<th>TB-P@k 10 Day</th>
<th>TB-MAP 10 Day</th>
<th>TB-P@k 10 Week</th>
<th>TB-MAP 10 Week</th>
<th>TB-P@k 20 Week</th>
<th>TB-MAP 20 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.206</td>
<td>0.393</td>
<td>0.308</td>
<td>0.424</td>
<td>0.358</td>
<td>0.469</td>
<td>0.297</td>
<td>0.472</td>
<td>0.305</td>
<td>0.508</td>
</tr>
<tr>
<td>Scoring Features</td>
<td>0.265</td>
<td>0.401</td>
<td>0.463</td>
<td>0.560</td>
<td>0.460</td>
<td>0.584</td>
<td>0.533</td>
<td>0.654</td>
<td>0.517</td>
<td>0.654</td>
</tr>
<tr>
<td>CB$^a$ Topic</td>
<td>0.396</td>
<td>0.476</td>
<td>0.526</td>
<td>0.584</td>
<td>0.566</td>
<td>0.628</td>
<td><strong>0.560</strong></td>
<td><strong>0.683</strong></td>
<td>0.526</td>
<td><strong>0.674</strong></td>
</tr>
<tr>
<td>UMA$^a$, UMS$^c$, Top 3</td>
<td><strong>0.432</strong></td>
<td>0.479</td>
<td><strong>0.611</strong></td>
<td><strong>0.605</strong></td>
<td><strong>0.650</strong></td>
<td><strong>0.668</strong></td>
<td>0.559</td>
<td>0.635</td>
<td><strong>0.566</strong></td>
<td>0.666</td>
</tr>
<tr>
<td>ST$^d$ &amp; LT$^e$ UM$^f$</td>
<td>0.396</td>
<td>0.476</td>
<td>0.526</td>
<td>0.584</td>
<td>0.566</td>
<td>0.628</td>
<td>0.560</td>
<td>0.683</td>
<td>0.526</td>
<td>0.674</td>
</tr>
<tr>
<td>Chb$^g$, UtM$^h$, Topic, Slope One</td>
<td>0.328</td>
<td><strong>0.530</strong></td>
<td>0.485</td>
<td>0.537</td>
<td>0.500</td>
<td>0.588</td>
<td>0.550</td>
<td>0.672</td>
<td>0.537</td>
<td>0.673</td>
</tr>
</tbody>
</table>

$^a$ Content-Based; $^b$ User Model Adaptation; $^c$ User-Model-Based Similarity; $^d$ Short-Term; $^e$ Long-Term; $^f$ User Model; $^g$ Collaboration-Based; $^h$ User to Message

Table 7.24: Results for various top time-binned measures for best of different approaches for Ratings without Interaction (RwoI).
<table>
<thead>
<tr>
<th>Run</th>
<th>$F_1$ Score</th>
<th>$F_2$ Score</th>
<th>TB-P@k 5 Day</th>
<th>TB-MAP 5 Day</th>
<th>TB-P@k 10 Day</th>
<th>TB-MAP 10 Day</th>
<th>TB-P@k 10 Week</th>
<th>TB-MAP 10 Week</th>
<th>TB-P@k 20 Week</th>
<th>TB-MAP 20 Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.180</td>
<td>0.354</td>
<td>0.305</td>
<td>0.409</td>
<td>0.374</td>
<td>0.465</td>
<td>0.272</td>
<td>0.498</td>
<td>0.297</td>
<td>0.518</td>
</tr>
<tr>
<td>Scoring Features</td>
<td>0.217</td>
<td>0.364</td>
<td>0.294</td>
<td>0.452</td>
<td>0.331</td>
<td>0.500</td>
<td>0.312</td>
<td>0.526</td>
<td>0.301</td>
<td>0.546</td>
</tr>
<tr>
<td>CB$^a$ Topic</td>
<td>0.332</td>
<td>0.430</td>
<td>0.503</td>
<td>0.562</td>
<td>0.531</td>
<td>0.605</td>
<td>0.483</td>
<td><strong>0.617</strong></td>
<td>0.464</td>
<td>0.627</td>
</tr>
<tr>
<td>UMA$^b$, UMS$^c$, Top 3</td>
<td><strong>0.377</strong></td>
<td>0.432</td>
<td><strong>0.583</strong></td>
<td><strong>0.606</strong></td>
<td><strong>0.612</strong></td>
<td><strong>0.649</strong></td>
<td><strong>0.509</strong></td>
<td>0.601</td>
<td><strong>0.556</strong></td>
<td><strong>0.642</strong></td>
</tr>
<tr>
<td>ST$^d$ &amp; LT$^e$ UM$^f$</td>
<td>0.331</td>
<td>0.430</td>
<td>0.504</td>
<td>0.562</td>
<td>0.532</td>
<td>0.606</td>
<td>0.481</td>
<td>0.617</td>
<td>0.463</td>
<td>0.627</td>
</tr>
<tr>
<td>ChB$^g$, UtM$^h$ Topic, Slope One</td>
<td>0.298</td>
<td><strong>0.494</strong></td>
<td>0.356</td>
<td>0.473</td>
<td>0.398</td>
<td>0.512</td>
<td>0.359</td>
<td>0.561</td>
<td>0.367</td>
<td>0.588</td>
</tr>
</tbody>
</table>

$^a$ Content-Based;  $^b$ User Model Adaptation;  $^c$ User-Model-Based Similarity;  $^d$ Short-Term;  $^e$ Long-Term;  $^f$ User Model;  $^g$ Collaboration-Based;  $^h$ User to Message

Table 7.25: Results for various top time-binned measures for best of different approaches for Ratings without Discussion Interaction (RwoDI).
Figure 7.12: Comparing different approaches for All Ratings (Rall).
### 7.11 Comparison of Different Approaches

<table>
<thead>
<tr>
<th>Measure Value</th>
<th>F1 Score</th>
<th>F2 Score</th>
<th>TB-P@k 10-Day</th>
<th>TB-MAP-10-Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.206</td>
<td>0.393</td>
<td>0.358</td>
<td>0.469</td>
</tr>
<tr>
<td>Scoring Features</td>
<td>0.265</td>
<td>0.401</td>
<td>0.460</td>
<td>0.584</td>
</tr>
<tr>
<td>CB^a Topic</td>
<td>0.396</td>
<td>0.476</td>
<td>0.566</td>
<td>0.628</td>
</tr>
<tr>
<td>UMA^b, UMS^c, Top 3</td>
<td>0.432</td>
<td>0.479</td>
<td>0.650</td>
<td>0.668</td>
</tr>
<tr>
<td>ST^d &amp; LT^e UM^f</td>
<td>0.396</td>
<td>0.476</td>
<td>0.566</td>
<td>0.628</td>
</tr>
<tr>
<td>CbB^g UtM^h Topic, Slope One</td>
<td>0.328</td>
<td>0.530</td>
<td>0.500</td>
<td>0.588</td>
</tr>
</tbody>
</table>

^a Content-Based; ^b User Model Adaptation; ^c User-Model-Based Similarity; ^d Short-Term; ^e Long-Term; ^f User Model; ^g Collaboration-Based; ^h User to Message

Figure 7.13: Comparing different approaches for Ratings without Interaction

[Rwol]
Figure 7.14: Comparing different approaches for Ratings without Discussion Interaction [KwoDI].
Chapter 8

Conclusion

In this chapter the results of this thesis are summarized and concluded. In Section 8.1 the main results of each of the previous chapters are summarized. In Section 8.2 the achieved results of this thesis are presented and explained. This includes the discussion of the research questions and theses. Finally, in Section 8.3 challenges and issues for future work related to this thesis are discussed.

8.1 Summary

In Chapter 2 the need of Enterprise 2.0 Social Media Stream Recommender (E2SR) was described through scenarios and use cases. Requirements were defined that are necessary for an E2SR. Two of the requirements deal with the quality of the recommendation: the first focuses on filtering all messages and the second on recommending the top relevant messages for each day or week.

Three research questions were formulated for an E2SR in Chapter 3 which deal with improving the quality of the recommendation. The first question deals with how such a recommender can learn without explicit user feedback. The second question considers how a recommender can adapt to new terms and interests. The third question reviews how short-term interest can enhance the quality of the recommender. For each research question a thesis was claimed.

In Chapter 4 related and previous work was discussed which also examined various short comings with respect to the requirements and the research questions. It showed that there are no recommenders that deal specifically with the characteristics of an E2SR and no one fulfills all requirements sufficiently.

According to the requirements and the research questions, algorithms which enhance the state of the art were developed in detail in Chapter 5. There, it is demonstrated how features can be extracted from messages...
and how they can be used to learn a term-based user model or to use a collaborative-based algorithm. An algorithm for user model adaptation is presented which detects new unknown terms in a user model and adapts them by looking at user models of similar users. Additionally, an approach is formulated which detects only the terms a user has interacted with during a certain period of time. Those are handled as short-terms and are only to be used for recommendation for a specific period of time.

The main aspects of the implementation are described in Chapter 6. The approaches were implemented within the open source framework SPEKTRUM which was created through the work of this thesis. Also, an integration to an existing E2SA was implemented.

Finally, the algorithms are evaluated against a dataset of an E2SA in Chapter 7. Four measures were endorsed that are used to compare the different algorithms: The $F_1$ and $F_2$ scores are used to measure the quality of the first requirement of filtering for messages. The $\text{TB-MAP}$ and $\text{TB-P@k}$ measures were developed to measure the quality of the second requirement to identify the top relevant messages for a time bin. Each of the approaches were evaluated independently and then the best of each approach were compared with each other. The results showed that a content-based topic-specific user model outperforms the other approaches and that the user model adaptation adds a noticeable quality increase to it when using similar users for adaptation.

### 8.2 Summarized Results

Based on the developed concept and the evaluation of the dataset, the following results have been achieved:

1. Extracting features from messages based on the interaction structure finds relevant results.
2. Using a content-based approach by employing a term-based user model for each topic separately increases the quality of the results, especially for finding messages when no previous interaction of the user is observable.
3. Learning the user model based on the extracted features leads to an E2SR that finds relevant results without using explicit user feedback.
4. Learning a user model for each topic separately is significantly better than learning only one user model.
5. A collaborative-based recommender that learns from the features only gives competitive results when dividing the messages into topics and running the collaboration filtering separately for each topic.
6. Adapting user models for unknown terms increases the quality of the recommendation.
8.2 Summarized Results

8.2.1 Research Questions

In Section 3.3, three research questions for this thesis have been formulated. A recommender concept has been formulated to research those questions in Chapter 5. In the evaluation (Chapter 7), this concept has been validated. Now, the questions are answered and the achievements are discussed.

**How can the quality of recommendation (measured by $F$-Scores and $TB-MAP$) be improved in an Enterprise 2.0 Social Media Stream Recommender without using explicit user feedback?**

The idea of extracting features based on interaction and discussion structure, as well as learning a content-based model leads to satisfactory results. The approach helps to identify relevant messages. A significant quality improvement has been made using a topic-specific user model ($F_1$ of 0.635 for $Rall$ and $TB-MAP$-10-Day of 0.605 for $RwoDI$). It is clearly shown that treating user interests independently based on topics is more successful than maintaining one global user model.

Also, the content-based approach outperforms the collaboration-based approach except for the $F_2$ measure for topic-specific $UtM$. It shows in both the best content- and collaboration-based approaches that the communications patterns are topic centered. It is also interesting that collaboration between user and terms does not add any value to the recommender and does not help at all to filter or find relevant messages.

**How can the quality of recommendation (measured by $F$-Scores and $TB-MAP$) be improved by adapting user models to new interests in an Enterprise 2.0 Social Media Stream Recommender?**

The approach of the user model adaptation noticeably increases the quality of the recommendation ($F_1$ of 0.650 for $Rall$ and $TB-MAP$-10-Day of 0.649 for $RwoDI$). The evaluation shows that it is crucial to carefully select the terms from which to adapt. This mainly addresses the selection of similar users and similar topics. Only the user-model-based on similarity of the top similar users was able to satisfy the expectations. Adapting too many terms leads to more messages with a higher relevance score. Again, this leads to a greater number of irrelevant messages being scored too high.

**How can a separation of short- and long-term interests improve the quality of recommendation (measured by $F$-Scores and $TB-MAP$) in an Enterprise 2.0 Social Media Stream Recommender?**

Surprisingly, the detection and integration of short- and long-term interests do not lead to an expected increase in the quality of recommendation ($F_1$ of 0.635 for $Rall$ and $TB-MAP$-10-Day of 0.606 for $RwoDI$). The detection
of short-term interest is based on the user model of the content-based approach. Therefore, the user model is not a perfect model of the users interest. Extracting the short-term interests of this imperfect model is then even more erroneous. This leads to an uncertain detection of such interests.

It is likely that, using only the features and interactions of a user, there is not enough information to detect and predict short-term interest. If the interaction for some terms is rare and sparse it is very hard or even impossible to detect any short-term behavior.

To overcome this, it must be further researched how this uncertainty can be narrowed using additional information. Such information can be obtained with more interaction feedback from the user.

8.2.2 Theses

Based on the research question three theses have been formulated (see also Section 3.3). Based on the discussion of the research question it is shown which theses are true and why.

**Thesis 1**

The first thesis is:

A content-based recommender with a topic-specific user model will lead to better recommendation quality than one with a global user model.

**Explanation**: As shown in the evaluation a recommender with a topic-specific user model clearly outperforms one with a global user model ($F_1$ of 0.635 for Rall and $TB-MAP_{10-Day}$ of 0.605 for RwoDI). Also, a collaborative-based approach ($F_1$ of 0.619 for Rall and $TB-MAP_{10-Day}$ of 0.512 for RwoDI) does not reach the quality of the content-based approach. Hence, **this thesis is true**.

**Thesis 2**

The second thesis is:

Adapting unknown terms in the user model by similar users or similar topics will lead to an increase in the quality of recommendation.

**Explanation**: The user model adaptation improves the quality in comparison with the best content-based approach when using a user-model-based similarity ($F_1$ of 0.650 for Rall and $TB-MAP_{10-Day}$ of 0.649 for RwoDI). Hence, **this thesis is true**.
Thesis 3

The third thesis is:

Separating short- and long-term interest will lead to an increase in the recommendation quality.

Explanation: In the recommender concept an approach was defined that maintains two user models, one with terms that are forgotten and one with steady terms (long-term). No significant improvement in the quality of the recommendation result was shown during the evaluation ($F_1$ of 0.635 for Rall and TE-MAP, 10-Day of 0.606 for RwoDI). As discussed in the research questions, it is not evident whether other approaches fulfilling this thesis will lead to an improvement or if the thesis itself is false. Hence, it cannot be stated that this thesis is true but it also cannot be stated that this thesis is false.

8.2.3 Dataset

It was not possible to find suitable and available datasets that can be used for further comparison. Due to the fact that enterprise data is by nature confidential it is hard to get access to different datasets. Also, for a recommender it is necessary to have a set of ratings for a user. No such dataset - known of and accessible to the author of this thesis - exists for enterprise recommenders.

In this thesis one internal dataset that was available has been used. The problem with one dataset is that optimization will tend to be dependent on the dataset or the domain of the dataset. The algorithm itself does not contain domain-specific or dataset-specific behavior so that the algorithm can be used in different domains as well. Of course the trained weights and configuration depend on the data, since the dataset is used to optimize them. It will be part of future work to evaluate the quality on other datasets and different domains.

8.2.4 Verifiability of Results

Due to the dataset issue, the implementation of the algorithm and evaluation is provided as open source in the SPEKTRUM[Com14] framework. This allows an easy reuse of the implementation and evaluation to do comparison studies for future work on different datasets or research areas.
8.3 Future Work

In this section several issues are discussed where future research could be done to further increase the quality and productivity of the described approaches.

8.3.1 Domain Patterns

In further research, the algorithm can be applied to other datasets of other domains. The dataset can then be used to optimize and evaluate the configuration as done with the dataset in this thesis. The results of the optimized configurations can then be compared to each other. If more datasets are available, a set of configurations can be analyzed to see if there are reoccurring or similar patterns in the configurations. For example, one result of this research could be that in one domain it is more feasible to learn from all messages the users interact with than in another domain with only some.

Once several of such patterns are known, an E2SR could give an administrator the option to choose or fine tune the recommender based on presets determined by the different patterns.

8.3.2 User Role Patterns

The idea of domain patterns can also be applied to different user roles. Typically, in an information technology company there is a project leader, consultant, software architect and developer. For each user, a role can be specified per topic. Then it can be analyzed to see whether there is a different configuration per user role. For example, for the project leader it might be more feasible to learn only from messages that the project leader is author of, but for the developer role more is learned from additional messages.

If user role patterns have been identified the next step is to assign a role (or multiple roles) to a user. This can be done directly by the user who selects his roles. Another idea is to let the user rate a few messages and compute how those ratings correlate with the user role patterns. As various roles are more confidently identified, the roles will be assigned to the user. The assigned roles will then influence future scorings and learnings for this user.

8.3.3 Semantics

Information extraction does not consider any semantics in the terms or use an ontology. Similar terms can be grouped and treated as one; an ontology can be used to find concepts behind the terms. For a usable recommender algorithm the ontology should be learned with zero or minimal user involvement (as in [Rao+13]). It would be impractical for a user to maintain an ontology by hand for each dataset.
8.3.4 Short- and Long-Term Interest Detection

As one conclusion, additional research can further discuss short- and long-term interest detection to find an approach that either leads to an improvement or to find reasons why short- and long-term interest separation does not work in this context.

One idea is to use emerging topic detection [CDC10] and apply it to terms. In that case, an emerging term is a term that occurs frequently in the current time bin but did not occur frequently in the previous time bins before. The emerging term score can then be used as weight in the user model.

For the long-term detection a separation can be necessary. If a term does not occur in a new user model time bin this can have two reasons: there is no new message in this time bin with this term or there is a new message but the user did not interacted with that message. In the second case it is clearly an indication that the interest of the user is missing especially if it happens in several time bins for the same term. If this happens the term should not be integrated into the long-term user model. In the first case, if the term does not occur again, either ignore the term also for the long-term model, or allow it in the long-term model until a new observation is made.

8.3.5 High Scalable Implementation

The current implementation of SPEKTRUM can be used for a system running on one node. The loose coupling of the components using the communicator gives the architectural option to run several instances of the algorithm on different nodes. To make the current implementation more scalable the persistence needs to be optimized, preferably using a NoSQL [Cat11] cluster capable of MapReduce [DG08] to transform database queries into different nodes. For the communicator, a general message queue implementation is necessary. If both are available and integrated into the framework it will be possible to run SPEKTRUM on several nodes and to scale horizontally.

8.3.6 Finding Common Interest

The UMA checks for an unknown term of one user model in other user models. As an alternative, terms of the user models can be analyzed to find groups of common terms. These are interest terms that occur together frequently. Such frequent patterns can be mined using the FP-Growth [HPY00] algorithm. For a user model it is possible to analyze which of the frequent patterns found exist almost completely, missing only one or two terms. The assumption is that the user model also contains a frequent pattern but not all terms have thus far been learned. Those missing terms can then be added to the user model. This is a foresighted UMA because terms are adapted in
advance without knowing if the term will occur again in a new message as part of the scoring process.

There are two challenges for useful adaptation based on common interests. One is to find the frequent patterns that are representative, not too specific (small) and not too generic (large). The other challenge is deciding when a frequent pattern partial matches a user model.

8.3.7 User Feedback

The recommendation algorithm is implemented within SPEKTRUM and it can be used by an E2SA such as Communote. A long-term user study could try to find out how useful the recommendations are as experienced by the users and not by the measures. This study should not be based on intensive ratings of messages by the user because it will be difficult to encourage users for several months. Instead, the users should be asked if they use the recommendation, if they think it is useful and if it helps to find information they would otherwise miss. During the study a user may rate messages the algorithm suggests to give a measure of usefulness of the recommendation.

The results of the study can be analyzed to see in which cases the recommendation should or shouldn’t be used. The study can also be used to identify possible user role patterns as discussed before.

8.3.8 Implicit Interactions

The recommendation approach that is used learns from message features. Further interactions can be used to learn from additional information. If it is possible to identify whether a user read or opened a message, this information can be integrated into the user model. Also, filtering or searching for certain topics or full text queries can be used to identify an interest of the user. A basic example of how such information can be identified is described in [Skr12]. Here, different observations of front-end interactions in Communote have been analyzed to see how they correlate with the relevance of messages. In most cases, a single front-end interaction is not enough to indicate an interest of the user in the object of the interaction. Therefore, in [Skr12] not until several interactions of one object within a short time frame have been observed can the object be used for learning. This approach can be integrated in the recommender algorithm described in this thesis to learn from front-end interactions.
Appendix A

Pre-Analysis

As one of the first steps in this thesis, an analysis of an E2SA was conducted. The goal of this pre-analysis was to obtain an understanding of the data and any change over time.

The pre-analysis is based on a dataset described in Section A.1. This dataset has been extended by ratings which the author of this thesis generated. The details and the insights of this self-rating evaluation are described in Section A.2.

Using the dataset, it has been analyzed in Section A.3 how extracted terms of the messages change over time. With the insights of the self-rating evaluation, in Section A.4 a similar analysis has been done learning a user model by simulated ratings. Again, the change of the user model over time has been analyzed.

A.1 Datasets

As a dataset the Communote data of the Communardo Software GmbH installation accessible by the author of this thesis has been used.

The Communote dataset used for this pre-analysis contained 74,316 messages, 240 topics and 132 users. The creation date for the messages in the dataset ranged from September 2008 to April 2012. During that time, Communote was introduced as a communication tool to Communardo Software GmbH. As a result, the number of created messages posted per month steadily increased during that time.

A.2 Self-Rating Evaluation

For the Communote dataset, no rating data was available as the pre-analysis was executed. In order to provide more insight, a part of the dataset has been rated by the author of this thesis.
A Pre-Analysis

For each message that has been presented to the rater (author of this thesis) during January 2012, an importance and an urgency had to be chosen. The importance rating has the following levels, starting with the one of highest importance:

**MUST_KNOW** The user must read this message.

**SHOULD_KNOW** The user should read this message.

**GOOD_TO_KNOW** The user does not necessarily need to read this message but it would be of interest to know.

**TOO_MUCH_INFO** The message might contain some interesting information but is by far not necessary to read for the user.

**NOT_OF_MY_BUSINESS** The message is not important for the user.

The distinction between **TOO_MUCH_INFO** and **NOT_OF_MY_BUSINESS** is small. The **TOO_MUCH_INFO** rating has mainly been used when the topic and main content are of slight interest, but not the message itself.

Independent of the importance, an urgency rating has been designated for each message according to the following levels:

**NOW** The message should be read now because action must be taken.

**DAY** The message should be read the same day that it was created.

**WEEK** The message should be read within the same week.

**MONTH** The message should be read within the same month

**YEAR** The message should be read within the same year.

**NEVER** The message will never be important for the user.

The ratings have been obtained by creating a simple web application that takes the messages as input. First, the user who will rate the message must be selected and then a set of messages will be presented. For each message a rating (importance and urgency) can be selected and will be stored in a database. The ratings have been generated for 3 users from the *Communote* dataset. The rater for all three users has been the author of this thesis, therefore the rater rated messages for himself and for the other two users. The rater knew the roles and tasks of the users and had enough insight to determine a nearly real rating.

Through this self-evaluation 2381 ratings of 1104 different messages for 3 users of the *Communote* dataset for January 2012 have been obtained. Although the users have varying permissions for the topics, most topics were accessible for each user used in the evaluation.

150
A.2 Self-Rating Evaluation

A.2.1 Self-Rating Dataset Analysis

In Figure A.1 the number of ratings for each importance is shown. The ratings can be summarized so that they indicate the following:

- 798 a high interest (MUST_KNOW and SHOULD_KNOW)
- 661 a normal interest (GOOD_TO_KNOW)
- 922 no interest (TOO_MUCH_INFO and NOT_OF_MY_BUSINESS)

Additionally, in Figure A.2 the urgency per importance is shown. The urgency for the no interest ratings is essentially not important: if the message has no relevance for the user it does not matter if it is urgent or not. For the other importance levels most of the messages are urgent within the same day or week but not later. Hence, the message should be handled quickly if it is relevant for the user. Only a few relevant messages have a lower urgency, all in the context of the self evaluation. As expected, if the importance is rated high then the message is urgent and vice versa.

A.2.2 Self-Rating Feature Analysis

For a single message and single user the following features have been used:

Author The user is the author of the message.
A Pre-Analysis

![Figure A.2: Self-Rating Evaluation: Distribution per importance and urgency.]

<table>
<thead>
<tr>
<th>Importance</th>
<th>Number of ratings</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT_OF_MY_BUSINESS</td>
<td>2</td>
</tr>
<tr>
<td>TOO_MUCH_INFO</td>
<td>4</td>
</tr>
<tr>
<td>GOOD_TO_KNOW</td>
<td>5</td>
</tr>
<tr>
<td>SHOULD_KNOW</td>
<td>9</td>
</tr>
<tr>
<td>MUST_KNOW</td>
<td>86</td>
</tr>
</tbody>
</table>

Mention  The user is notified in the message.

DiscussionStarter  The message is part of a discussion. The user is the author of the root message of the discussion.

Participation  The message is part of a discussion. The user is the author of another message within the discussion.

DiscussionNotification  The message is part of the discussion. The user has been notified in another message within the discussion.

Each feature can be easily computed from the given dataset. The statistics for each Feature are shown in Table A.1.

For each rating - that refers to a user and a message - the features can be easily computed. With this data a decision tree learner has been used to analyze and understand which features discriminated the dataset and their ratings most.

For learning a decision tree C4.5 [Qui93] has been used (Implementation in Knime [AG12] was used.). The main results from the learned decision tree are:
A.3 Term Analysis

<table>
<thead>
<tr>
<th>Feature</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>1173</td>
</tr>
<tr>
<td>Author</td>
<td>314</td>
</tr>
<tr>
<td>Mention</td>
<td>511</td>
</tr>
<tr>
<td>DiscussionStarter</td>
<td>214</td>
</tr>
<tr>
<td>DiscussionMention</td>
<td>318</td>
</tr>
</tbody>
</table>

Table A.1: Statistics for the features of the self-evaluation pre-analysis.

1. If the message is part of a discussion or the user started the discussion it is of high interest in 70% of the cases, 53% for MUST_KNOW and 18% for SHOULD_KNOW.

2. If the message is part of but not the root of a discussion and the user neither participated nor got a mention within the discussion, the message is of low interest for the user in 87% of the cases, 58% for NOT_OF_MY_BUSINESS and 29% for TOO_MUCH_INFO.

The results can be interpreted as follows:

- The discussion participation and notification features show a high discrimination between interest and no interest.
- The main focus for recommendation should be placed on the root message.

A.3 Term Analysis

In the second analysis, terms have been extracted from the messages and the distribution of the terms over different time spans have been analyzed. As datasets, all messages (not only from the self-rating dataset) have been used as described in Section [A.1](#).

The term extraction for a message is simple and straightforward:

1. Clean up the message - remove html tags and non-alphabetic characters
2. Determine the language
3. Split the message into single terms
4. Remove stop words (based on the extracted language)
5. Stem the terms (based on the extracted language)
No further filtering of the terms has been used (e.g., no filtering based on term frequency). The terms have been stored independently per topic. Technically, that means that term \textit{car} in topic 1 is not equal to the term \textit{car} in topic 2. The reason is that the following analysis should reflect the changes of terms independently per topic.

The messages have been arranged into several time bin distributions. Each distribution contains a set of bins with a precision and length. The precision defines how far two adjacent bins are shifted and the length defines the time span of the bin. For example, if the precision is daily and the length is monthly than the following bins are created:

- Bin $n$ contains messages from 06/27/2011 to 07/27/2011
- Bin $n + 1$ contains messages from 06/28/2011 to 07/28/2011

In this case the same message occurs in multiple bins. If the precision and length is the same, e.g. month, than the messages are split and occur in only one bin.

For term analysis the messages have been distributed to the bins. For a bin $n$ the following measures have been calculated:

- New terms: Number of terms that occur in bin $n$ but not in bin $n - 1$
- Removed terms: Number of terms that occur in bin $n - 1$ but not in bin $n$
- Matching terms: Number of terms that occur in bin $n$ and in bin $n - 1$
- Global new term: Number of terms that occur in bin $n$ for the first term, consequently did not occur in bin $x$ with $x < n$
- Global matching term: Number of terms that occur in bin $n$ and also occurred in bin $x$ with $x < n$

### A.3.1 Term Analysis Results

Results for the monthly bin calculations are shown in Figure [A.3](#). Here, for each month three values have been computed: the matching terms are the terms that occur both in the current month and the month before, the new terms are the terms that occur for the first time ever in the current month, and the removed terms are the terms that occurred in the last month but not in the current month. It is obvious that only a small amount of terms are matching; the rest are changing. There is a steady flow of new terms; when only comparing matching and new terms, more than 50\% of the terms are new for all months.

As expected in the beginning of the usage of \textit{Communote} most of the terms are new. But the rate of new terms falls slowly, because the messages contain new information described by new terms even after three years (36 months) of usage. A recommendation engine must be able to adapt and adjust to that new information.

154
A.4 User Model Term Interest Analysis

As the next step the user interests have been analyzed with the focus on the change in interest over time. The question to analyze is whether the user’s interests change in comparison to the term interest over time and how fast the change is. For this pre-analysis the Communote dataset has been used.

The main problem is how to obtain ratings that can be used to build a user model for recommendation. The solution for this analysis is to generate ratings based on the results of the self-rating evaluation (see Section A.2.2). The idea is to use the main user features as an indicator of interest or lack of interest for the message as follows:

1. MUST_KNOW if the user is the author of the message
2. MUST_KNOW if the user is mentioned in the message
3. SHOULD_KNOW if the message is part of a discussion and the user is the author or is mentioned within this discussion
4. TOO_MUCH_INFO if the message is part of a discussion and the user is neither the author nor mentioned within this discussion and the message is not a root message.

Through this method 4,877,136 ratings have been generated on the Communote dataset. About 300,000 of those ratings are positive ratings (for details see Table A.2). Based on the observation of the self-rating evaluation this rating generation simulates real world behavior. It will not replace a
A Pre-Analysis

<table>
<thead>
<tr>
<th>Dataset statistics for the discussion-based generated ratings.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of messages</strong></td>
</tr>
<tr>
<td><strong>Number of users</strong></td>
</tr>
<tr>
<td><strong>Number of generated ratings</strong></td>
</tr>
<tr>
<td><strong>Number of MUST_KNOW ratings</strong></td>
</tr>
<tr>
<td><strong>Number of SHOULD_KNOW ratings</strong></td>
</tr>
<tr>
<td><strong>Number of TOO_MUCH_INFO ratings</strong></td>
</tr>
</tbody>
</table>

Table A.2: Dataset statistics for the discussion-based generated ratings.

<table>
<thead>
<tr>
<th>Importance</th>
<th>Rating $r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUST_KNOW</td>
<td>1.0</td>
</tr>
<tr>
<td>SHOULD_KNOW</td>
<td>0.8</td>
</tr>
<tr>
<td>GOOD_TO_KNOW</td>
<td>0.6</td>
</tr>
<tr>
<td>TOO_MUCH_INFO</td>
<td>0.4</td>
</tr>
<tr>
<td>NOT_OF_MY_BUSINESS</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table A.3: Mapping of importance to rating values.

real world evaluation but it may provide indications and insights into the change of interest.

For the analysis a user model has been computed based on the Term Count Learning Strategy described in detail in Section 5.4. The importance ratings have been mapped to a numeric rating value as shown in Table A.3.

The goal of this analysis is to determine the change of interest terms over time. Therefore, the messages have been distributed into bins using the same principle as in section A.3. For each bin, user models have been computed. Based on this, the following measurements have been calculated:

- Number of matching interest terms: Number of interest terms that occur in the user models of the same user in both bins.
- Number of new interest terms: Number of interest terms that occur in the user models of the same user in bin $n$ for the first time ever.
- Number of removed interest terms: Number of interest terms that occur in the user models of the same user in bin $n - 1$ but not in bin $n$.

A.4.1 User Model Analysis Results

The results of the bin comparison are shown in Figure A.4, A.5 and A.6. Here, each bin contains messages for a month and each bin is shifted by a month (Figure A.4), day (Figure A.5) and week (Figure A.6). The daily and weekly shifted bins contain duplicated messages, hence the number of matching terms is significantly higher than in the monthly chart. Since the
same term can occur in multiple user models the overall term count is much higher than in the term analysis result.

The new and removed interest terms for the daily bins show the change during one day, one week and one month, respectively. It is interesting to notice that, similar to the term analysis, more than 50% of the interest terms change during a single month. This means that the overall user model changes by terms that are removed or added. The same applies for the daily and weekly changes. A steady amount of new terms occurs in the user model. In conclusion, this indicates that a recommender system that will score messages must be able to adapt within a week because interests can change heavily within a week and can even been turned around within a month.

This analysis did not account for long-term interest, a separation of slow and fast changing interest or of reoccurring interest. However, it is an indication of the need for an E2SR that handles changes over time.
A Pre-Analysis

Figure A.5: Self-Rating Evaluation: User model term change compared to previous bin of length month and each bin moved by one day.

Figure A.6: Self-Rating Evaluation: User model term change compared to previous bin of length month and each bin moved by one week.
Appendix B

Relevance Manifest

Prior to creating the dataset described in Section 7.2, a relevance manifest was written. This manifest was presented to each rater to provide a common understanding of what is relevance in the context of this thesis, that is, when is a message relevant for a user and when not. The relevance manifest was written in German by the author of this thesis and published in January 2013 in the internal wiki of Communardo Software GmbH. The original German version and the translation to English is presented next.

B.1 German Version

Was ist Relevanz?
Die Relevanz einer Nachricht drückt aus, wie relevant, bedeutsam, wichtig diese Nachricht für den Nutzer und seine tägliche Arbeit ist. Im Kontext des unternehmensinternen Einsatzes ist eine Nachricht für den Nutzer genau dann relevant, wenn die Nachricht eine Information enthält, die dazu führt, dass:

- der Nutzer oder ein Team effektiver arbeiten kann,
- der Nutzer oder ein Team effizenter arbeiten kann.

Im Gegensatz dazu ist die Nachricht dann irrelevant, wenn diese keine Information enthält, die zu einer Effizienzsteigerung führen.

Wie soll eine Nachricht bewertet werden?
In der Praxis ist es schwierig und der Übergang fließend, ob eine Nachricht relevant ist oder nicht. Für eine positive (relevante) Bewertung sollte überlegt werden:

- Hat die Information mir geholfen, schneller eine Aufgabe zu bearbeiten?
- Kann ich zu der Nachricht einen Beitrag leisten, so dass ein Teammitglied seine Aufgabe besser und schneller erledigen kann?
- Erhielt ich durch die Nachricht einen Erkenntnisgewinn der mir weitergeholfen hat?
• Hat die Nachricht Auswirkungen auf meine Arbeitsweise?
  – Z.B. Verschiebung von Prioritäten

• Wenn ich diese Nachricht nicht gelesen hätte,
  – fehlt mir diese Information?
  – hätte ich dann nachfragen müssen, um diese Information zu erhalten?
  – hätte ich wohlmöglich unnütz an einer Aufgabe gearbeit (wegen falscher Prio, nicht mehr notwendig, schon erledigt, etc)

Bewertung von selbstverfaßten Nachrichten:

• Hier sollte rein auf den Inhalt geschaut werden, und sich überlegt werden, ob der Inhalt für einen selbst interessant ist oder nicht. Dies sind i.d.R. Nachrichten die sich in dem Kontext bewegen, in dem sich die eigenen Aufgaben bewegen. Man schreibt aber auch Nachrichten, um anderen zu helfen, oder Tipps zu geben, die aber für die eigene Arbeit nicht relevant sind.

• Beispiel für eine Nachricht, die ein Autor positiv bewerten könnte:
  – ”@xyz hast du dich schon mit dem Cache Problem befaßt, hier sollten wir eine andere Cachestrategie verwenden”

• Beispiele für Nachrichten, die ein Autor negativ bewerten könnte:
  – ”danke dir!”
  – ”schau dir mal das SOAP Tutorial an, dass hatte ich in einem Projekt letzten mal angewandt und fand ich gut, mache damit aber jetzt nichts mehr.”


Die Bewertung einer Nachricht ist nur für einen Nutzer selbst sichtbar, gleiches gilt für den berechneten Score.

Zielstellung der Bewertung
Um gute Ergebnisse zu bekommen werden möglichst vollständige Bewertungen benötigt, das heißt das möglichst alle Nachrichten bewerten worden sind. Da die häufigsten Bewertung ein ’nicht relevant’ sein werden, ist es möglich aus den positiven (relevant) Bewertungen die negativen (nicht relevant) abzuleiten. Diese Schlussfolgerung kann aber nur dann gezogen werden, wenn die Nachrichten, die relevant sind, in einem Zeitraum auch abgeschlossen sind.
B.2 English Version

What is relevance?
The relevance of a message expresses how relevant or important this message is for a user and his daily work. In the enterprise context a message is relevant for a user if and only if the information of the message will result in:

- the user or the team working more effectively,
- the user or the team working more efficiently.

In contrast, a message is not relevant if it does not contain any information that will lead to an increase in efficiency.

How to rate a message?
In practice it is difficult to decide if a message is relevant or not. For a positive (relevant) rating the following should be considered:

- Did the information help to finish a task quickly?
- Can I contribute to the message such that a team member can finish his task faster?
- Did I receive a noticeable amount of new knowledge through this message?
- Does the message have an impact on my working behavior?
  - for example, changing of priorities
- If I had not read the message:
  - would I miss this information?
  - would I have to ask someone to get this information?
  - would I have worked unnecessarily on a task (due to incorrect prioritization, no longer being needed, already having been solved, etc)

Rating of self written messages:

- Here, only the content of the message should be considered, whether the content is relevant to yourself or not. Usually these are messages which focus on the context of your own tasks. A negative example is messages that are written to help someone else or to give hints which are not relevant to your own work.
- Example for a message to rate positive by the author:
  - "@xyz did you already discuss the caching problem? We should try a different cache strategy."
- Examples for a message to rate negative by the author:
  - "thank you!"
  - "take a look a the the SOAP tutorial. I used it in a project recently and it looked pretty interesting but I do not use it anymore."
**Summarized**: Do I want to have the message occur in my message stream or not? If yes, then that message should be rated **positive**. It should **not** be rated whether the message is good or bad. The Like function should be used for that.

There are three rating options:

- **Top Arrow**: The message is relevant to me.
- **Circle**: It is hard for me to assess if the message is relevant to me but I also cannot tell if the message is not relevant to me.
- **Bottom Arrow**: The message is not relevant to me.

The rating of a message is only visible for yourself, the same applies to the compute relevance score.

**Goal of the Evaluation**

To achieve good results, it is necessary to have mostly complete ratings. If possible, all messages should be rated. Because the most frequent rating will be negative (not relevant) it is possible to derive the negative ratings from the posting ones. This conclusion can only be drawn for messages within the time frame where all relevant messages have been rated positive.

**B.3 Remark**

In the goal of the evaluation of the manifest the idea has been formulated to derive negative ratings if all positive ratings have been formulated. Due to the amount of messages that have the possibility of being rated, this never took place. Only ratings submitted by the user have been used. No ratings have been inferred.
References


References


References


References


References


[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. “Mining frequent patterns without candidate generation”. In: SIGMOD Rec. 29.2 (2000), pp. 1–12 (cit. on p. 147).


References


References


References


