DRESDEN UNIVERSITY OF TECHNOLOGY

User Model Representations for Information Prediction

by

Ramzi Youssefi

Supervisors:
Dr.-Ing. Marius Feldmann
Dipl.-Medien-Inf. Philipp Katz

Professor:
Prof. Dr. rer. nat. habil. Dr. h. c Alexander Schill

July 2011

Declaration of Authorship

I, RAMZI YOUSSEFI, declare that this thesis titled, 'USER MODEL REPRESENTATION FOR INFORMATION PREDICTION' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Be generous in prosperity, and thankful in adversity. Be worthy of the trust of thy neighbour, and look upon him with a bright and friendly face. Be a treasure to the poor, an admonisher to the rich, an answerer of the cry of the needy, a preserver of the sanctity of thy pledge. Be fair in thy judgement, and guarded in thy speech. Be unjust to no man, and show all meekness to all men. Be as a lamp unto them that walk in darkness, a joy to the sorrowful, a sea for the thirsty, a haven for the distressed, an upholder and defender of the victim of oppression. Let integrity and uprightness distinguish all thine acts. Be a home for the stranger, a balm to the suffering, a tower of strength for the fugitive. Be eyes to the blind, and a guiding light unto the feet of the erring. Be an ornament to the countenance of truth, a crown to the brow of fidelity, a pillar of the temple of righteousness, a breath of life to the body of mankind, an ensign of the hosts of justice, a luminary above the horizon of virtue, a dew to the soil of the human heart, an ark on the ocean of knowledge, a sun in the heaven of bounty, a gem on the diadem of wisdom, a shining light in the firmament of thy generation, a fruit upon the tree of humility.

From The Baha'i Writings.

Acknowledgements

In this opportunity, I would like to thank the following people who made this work possible. My supervisor, Marius Feldmann, who helped me from the beginning of this master program at the TU-Dresden. I really appreciate everything that you did for me Marius, you helped me getting the Studentischehilfskrafts, and along the process of the supervision of the master thesis, thank you very much for your unlimited patience towards me. Philipp Katz, the other supervisor, who helped me with this master thesis with caring support and advise, I really took into account our discussions and mutual feedback. To my good friend Ricardo Riquelme, who helped in the process of writing this thesis. I truly appreciate the long time we spent correcting this work. Finally, I would like to thank some of my friends who helped with the evaluation: Anna Utlik, Khaled Reza, Jose Irigon, Chen Xi, Ivan Gudymenko, Jenn Janesko, Uwe Thomas and Thorsten Krueger.

Contents

Declaration of Authorship **iii**

Acknowledgements **v**

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Goals . . . 2
1.3 Requirement Analysis . . . 3
1.3.1 Use Cases . . . 3
1.3.2 Specific Goals . . . 4
1.4 Structure of the Work . . . 5

2 Background and State of the Art **7**
2.1 Background Information . . . 7
2.1.1 User Models . . . 7
2.1.2 Recommender Systems . . . 8
2.1.3 Content Based Filtering . . . 8
2.1.4 Collaborative Filtering . . . 8
2.1.4.1 Memory-Based Algorithms . . . 8
2.1.4.2 Model-Based Algorithms . . . 9
2.1.4.3 Hybrid Algorithms . . . 9
2.1.5 Mixed Collaborative and Content-Based Filtering . . . 9
2.1.6 Cosine Similarity . . . 9
2.1.7 Naive Bayes Classifier . . . 10
2.2 State of the Art . . . 10
2.2.1 Conceptual Recommender System for CiteSeer . . . 10
2.2.2 Google News Personalization: Scalable Online Collaborative Filtering . . . 12
2.2.3 PersoNews . . . 14
2.2.4 OmniSeer: User Modelling for Intent Prediction in Information Analysis . . . 16

2.2.5 The Mindful Reader . . . 18
2.2.6 Further Approaches . . . 20
2.3 Summary . . . 22

3 Description of the Concept 23
3.1 Recommender System Architecture . . . 23
3.2 User Interests Representation by Extended User Model . . . 25
3.2.1 Basic User Model . . . 25
3.2.2 Complementary User Model . . . 26
3.2.3 Context Vector Model . . . 27
3.2.4 Common User Model . . . 27
3.3 Extended User Model Generation . . . 29
3.3.1 Phase 1: Creation of the Complementary User Model . . . 29
3.3.1.1 Analysis of the Basic User Model . . . 30
3.3.1.2 Rank Calculation . . . 30
3.3.1.3 Domain Words Model Analysis . . . 31
3.3.2 Phase 2: Correlation Evaluation . . . 32
3.3.3 Phase 3: Adaptation . . . 32
3.3.3.1 Short-Term Preferences . . . 33
3.3.3.2 Long-Term Preferences . . . 33
3.3.4 Phase 4: Collaborative Filtering . . . 34
3.3.4.1 Cosine Similarity Evaluation . . . 35
3.3.4.2 Common User Model . . . 35
3.4 Recommendations . . . 35
3.4.1 Recommendations over Complementary User Model . . . 36
3.4.2 Recommendations over Context Vector . . . 36
3.4.3 Recommendations According Collaborative Filtering . . . 36
3.4.4 Combined Final Recommendation . . . 37
3.5 Summary . . . 38

4 Implementation 39
4.1 Development Languages . . . 39
4.2 Selected Aspects of the Implementation . . . 40
4.2.1 Graphical User Interface . . . 40
4.2.2 Database Model . . . 41
4.3 Packages Diagram . . . 43
4.3.1 Profilecreator Package . . . 43
4.3.2 Recommender Package . . . 45
4.3.3 Wordsdomain Package . . . 45
4.4 Extended User Model Instantiation . . . 46
4.4.1 Creation of the Basic User Model . . . 46
4.4.2 Creation of the Complementary User Model . . . 46
4.4.3 Creation of the Context Vector Model . . . 47
4.4.4 Collaborative Filtering Analysis . . . 47
4.4.5 Validity Control According to Timestamp . . . 48
4.5 Recommendation Analysis . . . 48
4.6 Summary . . . 51

5 Evaluation **53**
5.1 Description of the Evaluation 53
5.2 Objective Evaluation 54
5.2.1 Evaluation of the Basic User Model 55
5.2.2 Evaluation of the Extended User Model 56
5.2.2.1 Complementary User Model 56
5.2.2.2 Context User Model 57
5.2.2.3 Common User Model 59
5.2.2.4 Extended User Model 60
5.2.2.5 User Models' Comparisons 61
5.3 Real Users' Evaluation 63
5.4 Summary 65

6 Epilogue **67**
6.1 Conclusion 68
6.2 Future Work 70

Bibliography **71**

List of Figures

1.1 Use cases identified in this work. 3

2.1 Architecture of CiteSeer. 11
2.2 System components of Google news personalization. 13
2.3 Architecture of PersoNews. 15
2.4 Architecture of OmniSeer. 17
2.5 Document vector generated by the Mindful Reader without black list removal. 19
2.6 User model representation for movies recommendation. 21

3.1 Recommender system architecture utilized in this work. 24
3.2 Extended User Model for user interests representation. 26
3.3 Representation of the Domain Words Model. 26
3.4 Vector representation for context evaluation of the message *m1*. 27
3.5 User model representation between two users with common interests. . . . 28
3.6 Cluster representation of Common User Model. 28
3.7 Binary representation of a Common User Model. 28
3.8 Diagram flow of the phase "creation of the Complementary User Model". 29
3.9 Timestamp representation regarding lifetime and positioning. 34
3.10 Example of timestamp applied to Context Vector Model. 34
3.11 Recommendations according to their accuracies. 38

4.1 Graphical user interface navigation. 40
4.2 Relational Model of the recommender system. 43
4.3 Java package diagram. 44
4.4 Cosine Similarity and Naive Bayes Classifier regarding to their user models and use. 49

5.1 Data analysis of the recommendations according to the Basic User Model. 56
5.2 Data analysis of the recommendations according to the Complementary User Model. 57
5.3 Data analysis of the recommendations according to the Context User Model. 58
5.4 Data analysis of the recommendations according to the Common User Model. 59
5.5 Data analysis of the recommendations according to the Extended User Model. 60
5.6 Data analysis of the recommendations according to the Complementary User Model. 62

List of Tables

2.1 User model comparisons and fulfilled requirements. Legend: 1 - CiteSeer. 2 - Google news. 3 - PersoNews. 4 - OmniSeer. 5 - Mindful Reader. . . . 22

3.1 Summary of the presented user models. 38

5.1 Evaluation results of the Basic User Model with the creation of the user model considering 10% to 100% of classified articles. 55
5.2 Evaluation results of the Complementary User Model with the creation of the user model considering 10% to 100% of classified articles. 57
5.3 Evaluation results of the Context User Model with the creation of the user model considering 10% to 100% of classified articles. 58
5.4 Evaluation results of the Common User Model with the creation of the user model considering 10% to 100% of classified articles. 59
5.5 Evaluation results of the Extended User Model with the creation of the user model considering 10% to 100% of classified articles. 60
5.6 Comparison 10% to 100% of classified articles. 61
5.7 Average and accuracy of according to users' criteria. 63

Chapter 1

Introduction

In many electronic commerce companies, recommender systems are an important tool that allow users to make better decisions when buying a product or helping a company to evaluate their clients' interests. One of the main parts of the structure of a number of recommender systems is the user model. The purpose of the user model is to represent information related to the user according to his or her needs, taste or areas of interest. However, since users' preferences are not something that can be easily represented due to their constant changes, many recommender systems have been proposed with different strategies to identify users' needs and to model them in an innovative fashion.

Unfortunately, according to some studies like [10] and [11], many recommender systems are able to recommend only a certain number of items, such as feed messages, online products or emails. These items have already been sorted out and classified by companies in order to later recommend them to the users according to their interests. Another problem, according to [11], is that current user models are not able to represent users' interests when these have changed after a period of time; a situation that makes the recommendation ineffective.

This thesis proposes a different approach to interpret users' interests and at the same time a more effective way to represent their interests. The representation will be achieved by the use of the so called Extended User Model. As a theoretical framework, this user model will be contrasted with another user model proposed by a previous study.

1.1 Motivation

Nowadays, where users are exposed to a constant, changing and always growing amount of information, intelligent computer-based techniques, such as recommender systems, are attempting to deal with the problem of information overload classification. These recommender system tools can be used to provide personalized services in most e-business domains, benefiting both the customer and the merchant. The customer will be benefited by suggestions provided by a recommender system whose recommendations will reflect the users' interests. The increase takes place when customers are offered a wider variety of related items that are not normally included on a regular basis [10]. Another advantage of recommender systems is that they can also be used to find interesting information in emails, RSS feeds or even more so, in specialized platforms where users need to be notified about an interesting event.

Current recommender systems are not able to efficiently recommend users with various items; because they are only able to suggest a limited number of already classified products or sorted out information. Another problem, according to [10], is that current recommender systems are usually unable to recognize the implicit association among products, which have different names, but still refer to similar objects or a family.

The motivation of this thesis is to efficiently solve the problems mentioned previously in a improved way. In order to determine the efficiency of this proposal, an evaluation and a comparison will be carried out to contrast the novel approach with existing user models.

1.2 Goals

Along this project, three goals have been identified. The first goal is to create and develop a generic framework in order to compare and evaluate different user models. The second goal is to determine the impact of factors that are embedded into different user models to highlight their characteristics. Finally, the third goal is to analyse the results of the user models representation by combining their features to identify the advantages and disadvantages of this aggregation.

1.3 Requirement Analysis

This chapter presents the use cases of the system in order to have a better understanding of the requirements that have been identified to develop this work. These requirements will be presented in detail in the next sections.

1.3.1 Use Cases

Let us consider a user who is interested in receiving information that he or she may like, this user must be provided with a web interface where he or she can visit and receive the information. In the *main page* the user has the option to register and log into the system once he or she has registered. In the *configuration page* the user has to add one or many URL feeds to start receiving news or notifications. In this step, the user is expected to have a minimum interaction with the system. By considering the user's feedback, the user model is created and improved according to the information that he or she has classified. Once the user has classified the feed messages, the user can select the option *load recommended information*, and the recommender system will suggest information according to what he or she likes or dislikes. At the same time, the user expects to have his or her recommendations updated by the recommender system according to his or her new interests. Figure 1.1 shows a use case diagram for this system.

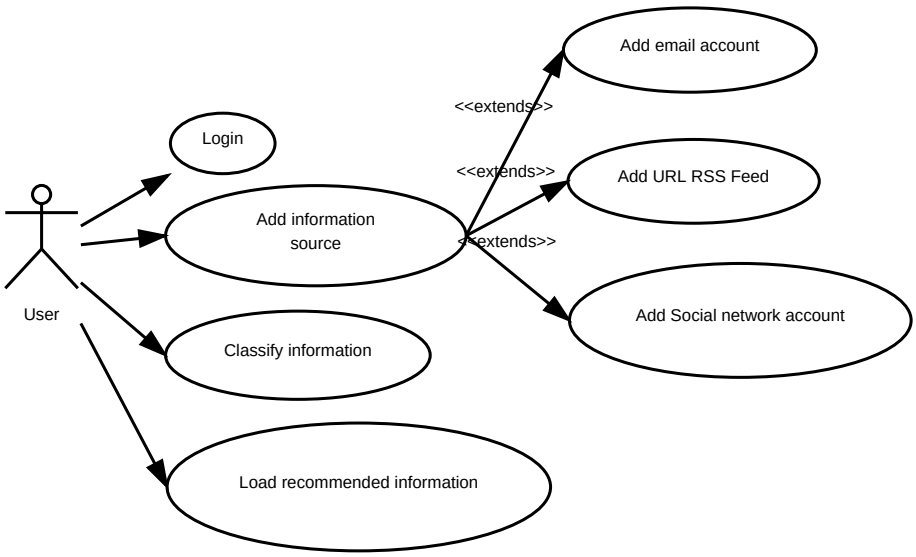

FIGURE 1.1: Use cases identified in this work.

1.3.2 Specific Goals

This work focuses on the development and analysis of user models. In order to predict information the user may be interest in, the user model interacts with the recommender system. Some of the current pieces of work like [2] are oriented in a specific task such as digital libraries, and other implementations like [7], which has another approach, recommend movies from a limited set of titles. In many cases, recommender systems consider the recommended information as an item to suggest rather than a thorough analysis of the user's interests. These items can even be rated with prior knowledge to evaluate whether the item is recommendable or not. This basic approach is simple if we intend to predict and recommend information that has not been previously classified. In other words, how can a recommender system predict and suggest articles or news to a user if this system does not know why the user likes or dislikes what he or she has classified? How can a recommender system analyse and recommend unrated items that do not belong to a determined set?

Because of what has been presented above, this work proposes two main concepts: a framework to analyse and evaluate user models and a user model that is capable of understanding and representing the preferences of a user according to its content and the context of a given article. In order to predict and recommend information the following requirements must be fulfilled for the proposed concepts:

1. **Supportability for a number of user models:** The purpose of the framework is to compare and evaluate different user models in order to determine their characteristic. Therefore, a framework that is capable to support various user models is desired.

2. **Unclassified items recommendation:** The user models must be able to predict and recommend information that has not been previously rated or classified. In other words, the user models must represent users' preferences to let the recommender predict information out of the set.

3. **Related items recommendations:** The user models should be able to represent users' interests in such way to let the recommender system predict alternative items that are implicitly associated to what the users like.

4. **Domain text extensibility:** The user model must be able to represent any kind of information that contains semantic terms, such as emails or news.

5. **Flexibility:** Flexible user models ensures the learning of the users' interests, as long as these users provide feedback. The recommendations must be more precise, as long as the user model learns the users' interests in an adequate manner.

6. **Adaptability:** In this concept, the user models must have a mechanism to identify changes of user's preferences as soon as he or she starts to show interest in other topics after some period of time.

7. **User model differences:** Identify the different evaluation results of the user models by comparing and analysing them in the same conditions and in the same evaluation set.

8. **User models suitability:** Once that the user models are compared and analysed, is important to determine the factors and the conditions that make a user model suitable for a specific case.

1.4 Structure of the Work

This work is structured in the following way: Chapter 2 defines terms and definitions used in this work and, at the same time, it presents the state of the art of user models and the recommender system. Chapter 3 describes the contributed concepts of this work, i.e. the necessary framework and the user models. Chapter 4 presents the implementation of the user models concepts and the recommender system. Chapter 5 shows the evaluation results of the implemented concepts and finally, Chapter 6 explains the drawn conclusions and future works.

Chapter 2

Background and State of the Art

This chapter provides an overview of existing solutions to predict information, and how user models can be represented. The goal is to acquire prior knowledge of current technologies to later compare these technologies with the approach proposed in this work. The first section presents a set of definitions related to recommender systems and its approaches. The second section introduces current works regarding this fields and finally, in the last section, a table will be used to summarize the comparison of the approaches.

2.1 Background Information

2.1.1 User Models

A user model is a set of data associated to a specific user. It can be used by a recommender system in order to personalize human-computer interaction and provide recommendations to a user according to how the user model has been created. To create a user model appropriately, it is very important to obtain the user's preferences using some kind of feedback, because the users' interests might change and the user model must learn the new interest of the user. This feedback can be explicit or implicit. Explicit feedback is collected directly from users through ratings and reviews. Implicit feedback is collected by implicit user actions like clicks, zooming image, basket insertions and purchases such as in e-commerce [6].

2.1.2 Recommender Systems

A recommender system attempts to analyse the user model to predict any interesting information that a user has not yet considered. It can be applied to items such as movies, TV shows, books, news or images. Recommender systems can be categorized in two types: Content based and collaborative filtering [5].

2.1.3 Content Based Filtering

Content based filtering calculates the similarity of the items defined in terms of their content, to other items that have been rated by the user to recommend new items [5]. One strength of this approach is that as long as the system has some information about each item, recommendations can be made even if the system has received a small number of ratings. The disadvantage of using this approach is that each item must be characterized with respect to the features that appear in the user's profile and, furthermore, the profile of each user must be collected and modelled [8].

2.1.4 Collaborative Filtering

Collaborative filtering is the process of analysing the interests of a user by collecting taste information from many users. In this approach, users are grouped into neighbourhoods according to a similarity metric and then used to recommend items to other members of the same cluster. For example, in a situation where a user A and B share the same interests, the recommender system will suggest both of them with similar items. Collaborative filtering can be categorized into two types: Memory based and model based algorithms [5].

2.1.4.1 Memory-Based Algorithms

Memory based algorithms make predictions for user based on their *past* ratings. According to the collaborative filtering definition, the prediction is calculated as a weighted average of the ratings given by "other" users where the weight is proportional to the similarity between users. To calculate this similarity between vectors, e.g. the Cosine similarity is implemented [9]. The popularity of this algorithm has been grown due to its simplicity, however, its scalability decreases when handling with longer texts and items.

2.1.4.2 Model-Based Algorithms

This model finds patterns to make predictions for users based on their past ratings and use these models to predict the ratings on *unseen* items using probabilistic methods. The advantages of this paradigm is that it handles the sparsity better than memory based algorithms. This helps with scalability with large data sets. The drawback of this model is that it is computationally expensive and, because of that, it generates a trade-off between performance and scalability.

2.1.4.3 Hybrid Algorithms

A number of works like [5] combine the memory-based and model-based algorithms. This approach improves the prediction performance by avoiding the limitations of both models above. Nevertheless, the complexity is increased and it is also expensive to implement.

2.1.5 Mixed Collaborative and Content-Based Filtering

A recommender system is described in [8] which uses a combination of content-based and collaborative methods to suggest items of interest to a user. This approach uses similarity measures among users, and it also measures directly the attributes of items that will make them appealing to specific users. In the work mentioned above, according to their obtained results, it is described that this method allows accurate recommendations for a sub-population of users, but not for the entire user population.

2.1.6 Cosine Similarity

Since that the created user model is a list of terms, it can be considered as a vector. A good approach to measure the similarity between two vectors is the Cosine similarity. The Cosine similarity is a common vector-based-measure where the input string is transformed into a vector space. In this vector space, the Euclidean cosine rule can be used to determine similarities. The Cosine similarity is described as follows:

$$similarity = \cos(\Theta) = \frac{AB}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Some of the main advantages of this method are its simplicity, speed and easy operability. However, when comparing two documents with similar number of words, the larger the texts, the smaller the similarities.

2.1.7 Naive Bayes Classifier

The Naive Bayes algorithm is a classification based on Bayes rule, that assumes attributes $X_1, ..., X_n$ that are conditionally independent of one another, given Y. The value of this assumption is that it dramatically simplifies the representation of $P(X|Y)$, and the problem of estimating it from the training data [13].

$$p(class_j) = \prod_{i}^{n} p(f_i|class_j)^N i$$

In text classification, the probability that an article falls into a class (interesting or non-interesting) is equivalent to the product of the individual probabilities for each informative term f existing in an article of that class, each raised in turn to the power of their overall frequency within the observed collection of documents N. The term i represents which feature f is presently being evaluated, out of n total features. The probability that an article falls into the "interesting" class can be used as a predicted interest rating [12].

The Naive Bayes Classifier is one of the most used method to classify documents. It is used to classify spams when the user has marked emails as "spam".

2.2 State of the Art

This section introduces some of the current recommender systems that fulfil a part of the presented requirements. Nevertheless, the focus of this section is to introduce the latest trend of the user model representations.

2.2.1 Conceptual Recommender System for CiteSeer

This work presented in [2] is a recommender system which suggests papers to users of the CiteSeer digital library. The user profiles are built according to the user's past-clicked history (implicit feedback). The system is content-based with both memory-based and model-based attributes.

The system creates a user profile for each user visiting the digital library, and it is used to identify the users' interests. Documents from the library have been parsed and classified to interact with the user profile. Once the user is logged in, The user profile contains the ACM[1] concepts the user is interested in ranked in descending order. The concepts in the user profile are used by the recommender system to suggest documents for the user. The system architecture of this work is described with the following diagram:

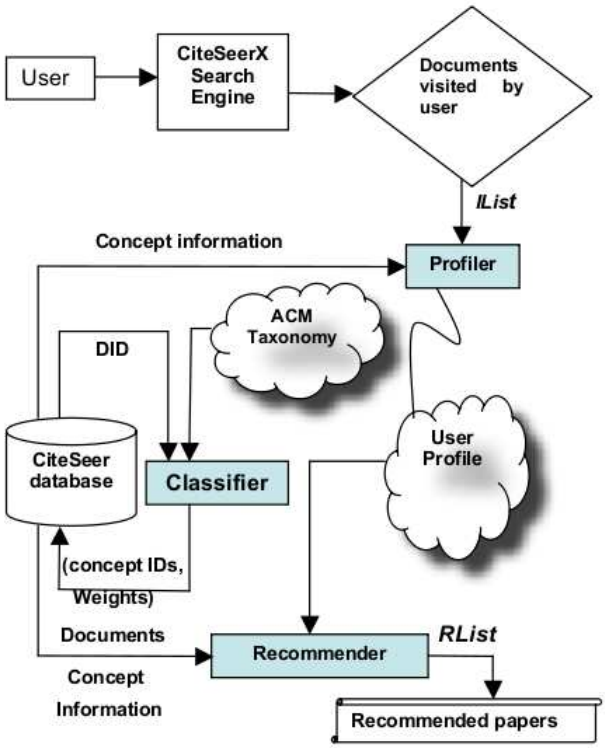

FIGURE 2.1: Architecture of CiteSeer.

The method used in this work to create the user model is by tracking, counting and storing the number of clicks of the documents visited by a user. This visit history and the document ID are the inputs to the profiler. After this, the top three concepts with their corresponding weights wt, for each of the documents visited by the user, are retrieved. In this case, wt represents a degree of association between a document and an associated concept as calculated by another module of the system called "categorizer". The user profile is created by using the three concepts and their weight information. Each concept with its corresponding weight pair is initially sorted according to their semantic meaning. If the user profile has more than one instance of the same concept with different weights, then these weights are added to compute the final weight associated with that particular concept in the user profile.

[1]Association for Computer Machinery.

wt*(cp,j)* = $\sum$ wt*(cp,i)*, for all documents i for the user j
where:
wt*(cp,j)* = weight of concept *cp* in user profile j
wt*(cp,i)* = weight of concept *cp* in document i

The final output of the profiler module is a list of concepts and their corresponding weights, that the user may be interested in, arranged in a decreasing order. The classifier does the categorization of the documents in two stages, training stage and classification stage, and it stores them in the CiteSeer database. The recommender module uses the user profile as the input. The output of this module is finally the recommended documents for the user.

Although, CiteSeer's recommender system is able to recommend a dataset collection of over 5,000,000 documents, new documents must be classified again in order to be recommended, this suggests that the recommender is not able to "predict" items out of the set. In this work, in order to create the user model, the system considers the concepts that the scientific papers contain. Nevertheless, the relationship between two or more terms is not described in any fashion. The recommender system does not support collaborative filtering, therefore, other user preferences are not utilized to improve the recommendations. Finally, this approach uses implicit feedback, that is to say, the user model does not consider what the user dislikes.

2.2.2 Google News Personalization: Scalable Online Collaborative Filtering

Google News Personalization is a collaborative filtering implementation to generate personalized recommendations for Google News users. The main goal is to develop a scalable system according to the users' interests and clicked-history of the community.
Google uses three approaches to develop this work: collaborative filtering using MinHash clustering [5], Probabilistic Latent Semantic Indexing (PLSI[21]) and co-visitation counts. One of the interesting features of this work is its domain independency, i.e., it could be potentially extended to other domains such as images, music and videos.

To achieve scalability, Google News clusters users, by using Locality Sensitive Hashing, to calculate the similarity between them. The key idea is to hash the input items so that similar items are mapped together in the same buckets with high probability.

Once that the users are clustered, Google News maintain the following statistics for each cluster at serving time: the number of clicks, decayed by time, that were received on different stories by members of this cluster. In case of PLSI[21] (statistical technique for the analysis of two-mode and co-occurrence data), the count of clicks is further weighted by the fractional cluster membership [5].

To see if a piece of news is recommendable for a user, Google News calculates and ranks unnormalized score-based news in clusters as follows: the system fetches the cluster from where the user belongs to and, for each cluster, the system counts the number of times that the members of a cluster have clicked on the news feed. After that, these numbers of clicks are added together to calculate the recommendation score as 0 to 1. These procedures are calculated separately by MinHash and PLSI.

Instead of utilizing only user-based recommendations, item-based techniques, such as co-visitation instances, are also implemented. Co-visitation is defined as an event in which two news are clicked by the same user within a certain time interval, typically set to a few hours. The co-visitation based recommendation score for a candidate is generated as follows: Google News fetches the user recent click history and it looks up the entry in the database by adding the value stored in this entry. This entry is normalized by adding the sum of all the entries of the user's click history. Finally, the co-visitation scores are normalized by linear scaling to a value between 0 and 1.

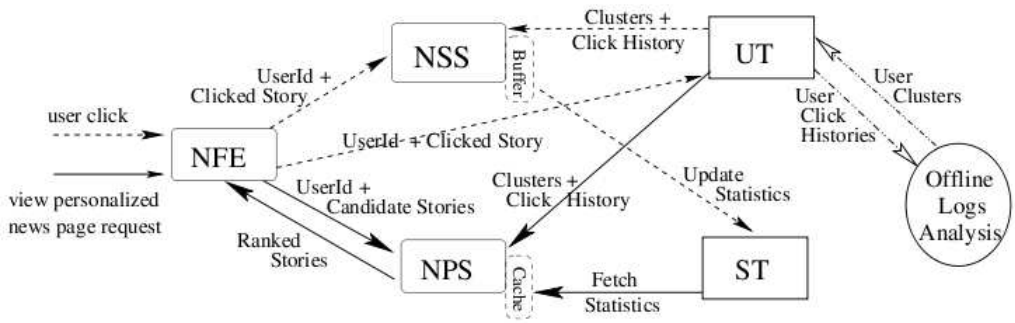

FIGURE 2.2: System components of Google news personalization.

After the recommendation system is set it requires the following three main components to perform:

- An offline component that is responsible for periodically clustering users based on their click history.
- A set of online servers responsible for performing two main types of tasks:

  - Updating user and story statistics each time a user clicks on a news story, and
  - Generating news story recommendations for a given user when requested;

- and two types of data tables:
  - a user table UT indexed by user-id that stores user click history and clustering information,
  - and a story table ST indexed by story-id that stores real time click counts for every story-story and story-cluster pair.

Figure 2.2 describes the components above explained. The abbreviations mean the following: **NFE:** News front end; **NSS:** News static server; **NPS:** News personalized server; **UT:** User table and **ST:** Story table.

The presented approach is very complex and is able to recommend to a large set of users. Nevertheless, it is not presented to recommend news that are out of the set. Google news personalization recommends news that belong to a set that the system has already acquired. This means that the user cannot add his or her own news feed. This approach does not fulfil the requirements to recommend alternative out of a set feeds. At the same time, since this approach is content agnostic, it does not consider either the content of the news nor the relationship between words. Finally, this work does not support user's feedback if he or she does not like a news feed article.

2.2.3 PersoNews

PersoNews is an enhanced RSS reader that is able to learn and classify interesting news for the user using a Naive Bayes Classifier. Some of its requirements are dealing with the information overload problem, a situation which is present more in the world wide web. In addition, PersoNews classifies general topics of interests such as computer science, medicine or education.

Users are provided and suggested with interesting articles from various sources. Some of these articles may appear in general Computer Science Journals or in a more specialized area of knowledge, such as scientific databased conferences or book reviews. At the same time, the user may be interested in commercial database management system such as ORACLE. Therefore, interesting articles can be found in ORACLE's RSS feeds. In PersoNews, sources are monitored under the same topic of interests, and additionally, a classifier personalizes the items gathered by each user.

PersoNews works with a classifier that has the following requirements[3]:

- An evidently good classifier for text categorizations tasks.
- The classifier must learn from feedback given by the user.
- In order to save computational cost, the classifier must be compact.
- A classifier must build dynamically the feature space as more documents/articles arrive.

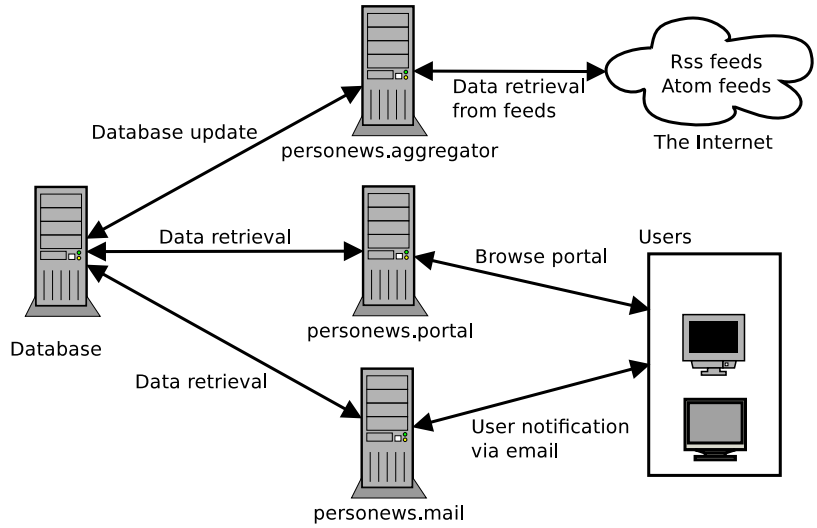

FIGURE 2.3: Architecture of PersoNews.

The architecture of PersoNews consist in three main modules:

- Website (PersoNews.portal): It is the interface between the user and the application.
- System update service (PersoNews.aggregator): It belongs to the server side and it monitors if there are new publications and it updates the database.
- Email notification service (PersoNews.email): It is responsible for notification if a feed has been updated or a new interesting feed has been published. It is also customizable.

One of the interesting features of PersoNews is that it allows the user to define one or more keywords. These keywords are extra filters for new publications, making them relevant if they have some of the keywords that the user has defined. When the user

starts monitoring a topic, PersoNews searches for subtopics and includes them in the keywords of the selected topic. PersoNews is one of the systems that can fulfil the requirements presented. The system is able to recommend feeds that are out of a set, in addition, it is flexible due to the explicit feedback provided by the user through the function like or dislike. However, users cannot share their preferences and therefore the collaborative filtering approach is not present.

2.2.4 OmniSeer: User Modelling for Intent Prediction in Information Analysis

Modelling for analyst intent prediction is a part of the OmniSeer project from the University of Connecticut and the University of South Carolina. The main focuses of OmniSeer are research, evaluation of technologies and exploration of algorithms or processes to develop system concepts that will assist the analysis of massive data. At the same time, OmniSeer is also able to make prior and tacit knowledge explicit allowing modification and update of the analyst's current cognitive state.

A primary objective of OmniSeer is to evaluate dynamic cognitive context models that support a comprehensive behavioural model such as user role, task preferences and interests. To explore models of analysts and their decision, reactive and autonomous tools are implemented. The following components form part of the architecture of OmniSeer:

- **User model component networks:** each user model of an analyst consists of three basic components:
  - Interest, what the analyst is focusing on.
  - Context, why the analyst focuses on the interest.
  - Preferences, how the analyst seeks and views information.
- **User model loading:** Used to initialize a user model at the start of an analyst's information seeking session.
- **User model updating:** Updates the user model based analyst's feedback and other services.
- **User model query:** Supports other services by providing information on analyst's current goals and interests.
- **User model explanation:** Provides detailed feedback to analysts and other services regarding decisions made by the user model and facilitates the exchange among analysts.

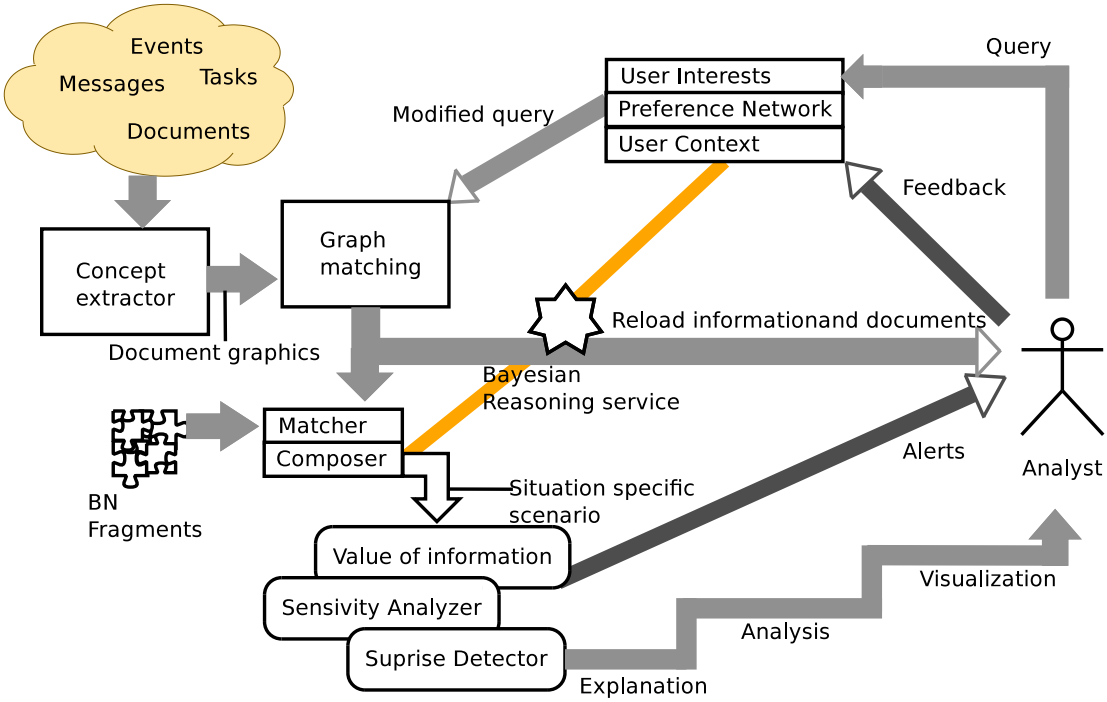

FIGURE 2.4: Architecture of OmniSeer.

- **Analyst query modification:** The Analyst's information queries are modified and new queries may be recommended to the analyst based on the current user model.

OmniSeer works as follows:

When a user starts the system, his or her user model is loaded by the User Model Loading module which recalls prior information about his or her information seeking behaviours such as his or her interests. First, he or she issues a query, which will be processed through the Analyst Query Modification component, to be modified according to what the system understands about the analyst's intent captured in the User Model Component Networks. Second, the modified query will be matched against every document gathered from events, tasks, messages and documents circulated within the organization. Each of these documents is processed through a Concept Extractor component where the concepts and the semantic relationships between them are extracted for more detail. Third, the analyst goes through the returned results and indicates which documents are relevant. This relevant information will be fed to the User Model Updating component where each of the component networks will be updated correspondingly. Finally, based on how the analyst assesses these recommendation, he or she will then provide feedback to the user model to update the user model accordingly [1].

To provide a fine-grained dynamic cognitive model and capture the analyst's goals and intentions, OmniSeer constructs a unified and dynamic model of the user's interest, preferences and context by keeping a track of his or her interests of the topics. Denote each interest concept as a and each level of interest for this concept as L(a). We compute L(a) after each query by L(a) = 0.5 (L(a) + n m) in which n, m are the number of relevant documents containing concept a and the number of relevant documents respectively. The user context network is constructed dynamically from the relevant documents.

The user preferences network, which represents how the user wants to modify the original query, is the last stage of the process. In addition to using the information contained in these components, to modify the analyst's query, the focus is on deriving and learning the analyst's working context by discovering the relationships which are not explicitly defined in the documents. This is done by gathering statistical information on linguistically related concepts. Moreover, as presented above, this approach is query oriented, and it has not been described any feedback from the user side. Recommended items belong to a set and no collaborative filtering has been implemented.

2.2.5 The Mindful Reader

The Mindful Reader is a project of the Worcester Polytechnic Institute, Massachusetts that centers on the design and development of a machine learning-augmented newsfeed aggregation application. It seeks to reduce the time necessary for users to find interesting newsfeed articles, by building a user interest model from implicit and explicit article ratings and applying that model to rank incoming articles based on predicted user's interest. The software was developed by using codes from the RSSOwl project; in tests, the user interest model grew more accurately over time.[12]

To identify the newsfeeds that are interesting for the user, the Mindful Reader uses implicit feedback by taking into account the variables like mouse clicks, scrolling and reading speed. These metrics are combined at the end of the article viewing to calculate the user's implied interest in the article. Next, the article itself is decomposed into a document vector consisting of terms and term frequencies.

The Mindful Reader also performs some additional processing steps at the document vectorization stage. Terms containing non-alphanumeric symbols such as pound signs and backslashes are dropped and all terms are reduced to lower case. Moreover, a predefined black list is used to remove words like:

Term	Frequency	Term	Frequency	Term	Frequency
the	5	to	3	article	1
mindful	1	reduce	1	ratings	1
reader	1	time	1	applying	1
project	1	necessary	1	that	1
centers	1	for	1	rank	1
on	2	users	1	incoming	1
design	1	find	1	based	1
development	1	interesting	1	predicted	1
of	1	articles	2	software	1
a	2	by	1	was	1
machine	1	building	1	developed	1
learning	1	user	3	using	1
augmented	1	interest	3	code	1
newsfeed	2	model	3	RSSOwl	1
aggregation	1	from	2	project	1
application	1	implicit	1	in	1
it	1	and	2	tests	1
seeks	1	explicit	1	grew	1
more	1	with	1		
accurate	1	time	1		

FIGURE 2.5: Document vector generated by the Mindful Reader without black list removal.

- Prepositions (on, of, from, etc.)
- Pronouns (he, she, it, they, etc.)
- Articles (a, an, the)
- Basic verbs (to be, to go, to do, etc.)
- Conjunctions (and, or, but, etc.)
- Numbers (one through ten and common resolution values)
- HTML markup (tags, URLs, etc)
- Variations on the above (contractions and abbreviations)

These words are removed because they do not provide extra information to distinguish between interesting and non-interesting articles. Regarding to the recommendations, the first method of calculating a predicted rating for a document involves constructing the rating as a weighted average of the known weights of each document term that also exists in the Informative Terms Database. The algorithm is as follows:

$$R_{AVG} = \sum_{i=0}^{n} \frac{weight(term_i)}{count_i}$$

where *weight(term)* provides the historical average weight of articles in which the term has appeared (obtained from the Informative Terms Database), $term_i$ provides the ith term in the document vector, $count_i$ provides the frequency of the ith term in the document vector, and n is the number of unique terms in the document vector.

The second method to predict recommended articles is a series of Naive Bayes classifiers. The algorithm is used to calculate probability that an article falls into a given class, by calculating the product of the probabilities of each term occurring in an article of that class. With the probabilities for each class calculated, these probabilities are then normalized so that they add to 1, and they are used to construct a rating. For the purposes of the math, it is convenient to label the five classes (from -2 to +2) as classes zero through four. Each classifier contributes a portion of the final rating, effectively voting on the outcome: the class 4 classifier contributes a portion rated at 1, the class 3 classifier contributes a portion rated at 0.75, and so on.

Finally, as a conclusion, in the Mindful Reader project, it has not been found any method to identify the context of a newsfeed nor words that are related to the preferred article by the user. In this work, any model that permits collaborative filtering between the users of the recommender system was not found

2.2.6 Further Approaches

Several works have proposed different algorithms for recommender systems. One example of them is [11] where they propose an effective way to calculate clusters of user models with collaborative filtering. According to [11] existing collaborative filtering algorithms do not consider the change of user interests. To solve this problem they proposed a "user-page" matrix to calculate the similarity between user interests. The algorithm considers a set of users and a set of web pages, and the time the user invests to visit every page. If the user stays longer on the web page, it means that he or she presents more interest in it. Using this determination the algorithm works as follows:

Input: user set $User = \{user_1, user_2, ..., user_m\}$; page set $URL = \{url_1, url_2, ..., url_n\}$;
Output: a user set with similar interest to $User_i : N(User_i)$.
Step 1: calculate user visit measure $R_{i,j}$ with user set $User_i$ and page set Url_j.
Step 2: construct "user-page" relevance matrix $M_{n \times m}$.
Step 3: calculate average score $\bar{r_c}$ for page Url_i.

Step 4: calculate $sim(i, j)$ with user $User_i$ and $User_j$ for the page Url_j.
Step 5: repeat step 3 and step 4 until calculate all interests similarity, and construct user set $N(User_i)$ who have similar interests with user $User_i$.
Step 6: return $N(User_i)$.

The algorithm generates set of users $N(User_i)$ and, using cosine similarity, it analyses the similarity between users clustering and users with more similarities.

In [7] proposes a collaborative recommender system for a defined set of items. In this work a set of movies is used as an example, and their goal is to identify users that belong to the same cluster. In order to do so, it presents a simple user model with the following structure:

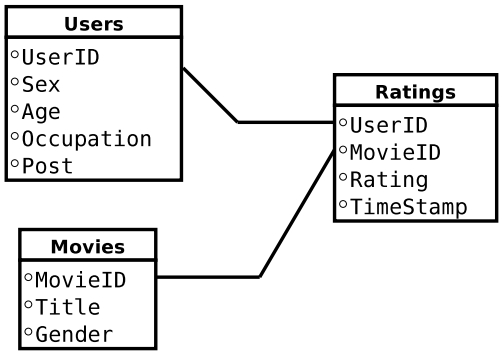

FIGURE 2.6: User model representation for movies recommendation.

This picture defines the interest of every user to a defined set of movies. The interest is represented according to a rank from 0 to 5.
The user model interacts with a collaborative filter called Taste. This framework implements some methods like **NearestNUserNeighbourhood** which returns a list of N nearest neighbours and **ThresholdUserNeighbourhood** which returns a list of neighbours that are greater than the established parameter.

In other works like [4], user models are presented by taking the user's preferences and creating his or her profile using a vector of linguistic utility, $P_u = \{p_1^u, p_2^u, ..., p_l^u\}$, where each p_i^u represents an attribute preference i of the user u. In order to get P_u, they use a set of criteria $C = \{c_1, ..., c_k, ..., c_l\}$ to describe an object according to the user interests. $A = \{a_1, ..., a_j, ..., a_n\}$ is the set of alternatives or products that can be recommended by the system, where $a_j = \{v_1^j, ..., v_k^j, ..., v_l^j\}$ and v_k^j is the value that is assigned to the product a_j with the criteria c_k. This value could be a tag or a number. $P_u = \{p_1^u, p_2^u, ..., p_l^u\}$ is the user profile from the user u where p_i^u is the value assigned by the user u according to a particular criterion c_i. This value is any tag provided by the system for that criterion.

Once the user model is represented, the product for each user is calculated according to his or her preferences using similarity measures.

2.3 Summary

These presented algorithms do not fulfill the requirements outlined in Chapter 1, *Introduction* either because they do not recommend alternative recommendations nor they recommend different types of text information. Moreover, none of them are able to recommend or predict information that is out of the set. As a conclusion, the following table is presented in order to have an overview of the recommender systems and their respective advantages and disadvantages:

Features / Work	**1**	**2**	**3**	**4**	**5**
Out of the set recom.	no	no	yes	no	yes
Text analysis	yes	no	yes	yes	yes
Context association	no	no	yes	no	no
Alter. items recommend.	no	no	no	no	no
Change of preferences	yes	yes	yes	no	yes
Dislike feedback	no	no	yes	no	no
User similarity	no	yes	no	no	no
Domain text extensibility	no	no	no	no	no

TABLE 2.1: User model comparisons and fulfilled requirements. Legend: 1 - CiteSeer. 2 - Google news. 3 - PersoNews. 4 - OmniSeer. 5 - Mindful Reader.

Table 3.1 shows how the requirements of this work may be fulfilled. PersoNews seems to be a very good approach to fulfil the requirements. However, it is not explained how the user model is created. Therefore, this work attempts to create a different user model. Unlike PersoNews, this model will be able to recommend items according to the information collected from other users' interests. This recommended information will have a certain degree of similarity among users' interests.

Chapter 3

Description of the Concept

This chapter explains the concepts of this work. It is described through the following sections: first, the introduction, which presents an overview of the system architecture that is necessary in order to use the Extended User Model and perform recommendations. Second, the Extended User Model is described as prior knowledge of what the model represents. Finally, the phases, that are necessary to create the Extended User Model and how the recommendations are performed, are presented.

The design of the concept has been driven by the requirements and research questions that were introduced in the motivation section of the introduction chapter. The implementation and evaluation of this concept is revised in the next chapters of this thesis.

3.1 Recommender System Architecture

As introduced in the first chapter, user models are a very important part of recommender systems, therefore, they can not be described without mentioning both concepts. An overview of the recommender system architecture is presented in figure 3.1. The recommender system is structured as follows:

- **Profile Creator:** In order to create a user model, the Profile Creator is the component that analyses what the user likes or dislikes. After the Profile Creator evaluates the user preferences, it creates a user model according to one of the algorithms in the user model definitions[1].

[1] Approach based on the Strategy Pattern.

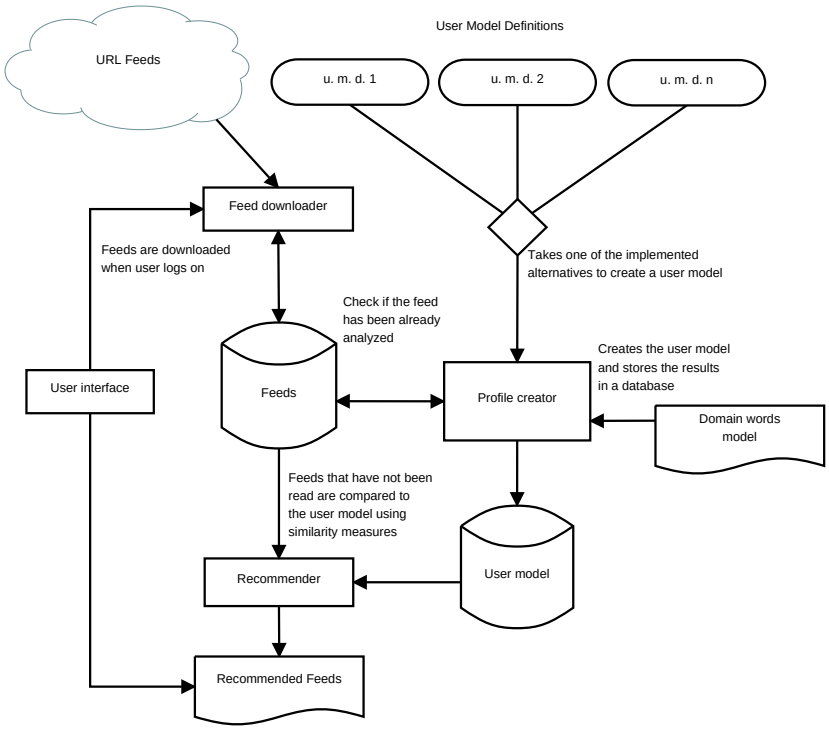

FIGURE 3.1: Recommender system architecture utilized in this work.

- **Domain Words Model:** One of the major weaknesses that has been identified during the discussion of example systems in Chapter 2, *Background and State of the Art*, is that the latent association among terms is not covered [10]. This makes it impossible to recommend messages that are implicitly related if they have different words. To solve this problem, a set of keywords has been considered, hierarchy related according to the context association between the words. This association makes easy to understand the relationship between words that belong to the domain and the messages that contain them. The Domain Words Model acts as a set of definitions that must be interpreted by the Profile Creator in order to generate a *Complementary user model*.

- **Recommender:** The Recommender interprets unclassified messages according to the Extended User Model (or any arbitrary user model definition) and evaluates whether the messages are recommendable or not. In order to do so, the Recommender evaluates which type of information must be analysed to use either a similarity or a probability measurement. A further explanation of the use of similarity and probability measurements will be provided in the following sections of this chapter.

- **User interface:** The user interface controls the user authentication. It allows the user add his or her own message provider (URL feed). It presents the messages to the user and shows the recommendations. When the messages are displayed, the

user has the chance to classify the information as likes or dislikes. This function enables the user to give explicit feedback to the system, which permits the user model to adapt to his or her interests.

As presented in the system architecture, the Profile Creator concretizes a user model definition, a set of rules and algorithms, to create a user model. The main contribution of this thesis is to present the Extended User Model as a model that is able to represent users' interests and later to compare the Extended User Model with other user model definitions.

3.2 User Interests Representation by Extended User Model

This section presents the Extended User Model, as one of the model definitions that interacts with the Profile Creator, in order to have a better overview of how the users' interests are represented by using this model definition. The purpose of the Extended User Model is to represent more broadly the users' interests. In addition to this, to allow the recommender system provide better recommendations. To fulfil this and other requirements presented in table 2.1, different aspects of users' interests have been evaluated as represented in figure 3.2. These aspects are defined as follows:

3.2.1 Basic User Model

The Basic User Model is the representation of the users' interests in a simple way. It contains a set of keywords with their frequency. In this set, the stop words have been removed and a stemming algorithm is used. These procedures are used, first, as a filter to increase the performance of analysis of the model, and second, to evaluate only the important words of the classified messages[2]. In this model, a timestamp is used to evaluate the validity of the words regarding the change of interests of the user after a period of time. The function of the Basic User Model is to provide the necessary information to the Profile Creator in order to create the Complementary User Model. This procedure is explained in section *Extended User Model generation.*

The Basic User Model can be represented as $U_i = ((w_1, f_z, t_{\bar{a}}), (w_2, f_y, t_{\bar{b}}), ..., (w_k, f_a, t_{\bar{d}}))$ where w is a word that belongs to a set of classified messages by a user i with frequency f, k is the number of words of the classified messages without repeated words and t is the timestamp of the word which is set when the word has been either added or updated.

[2]Stopword and stemming algorithm are used as a pre-processing phase every time that the messages are analysed.

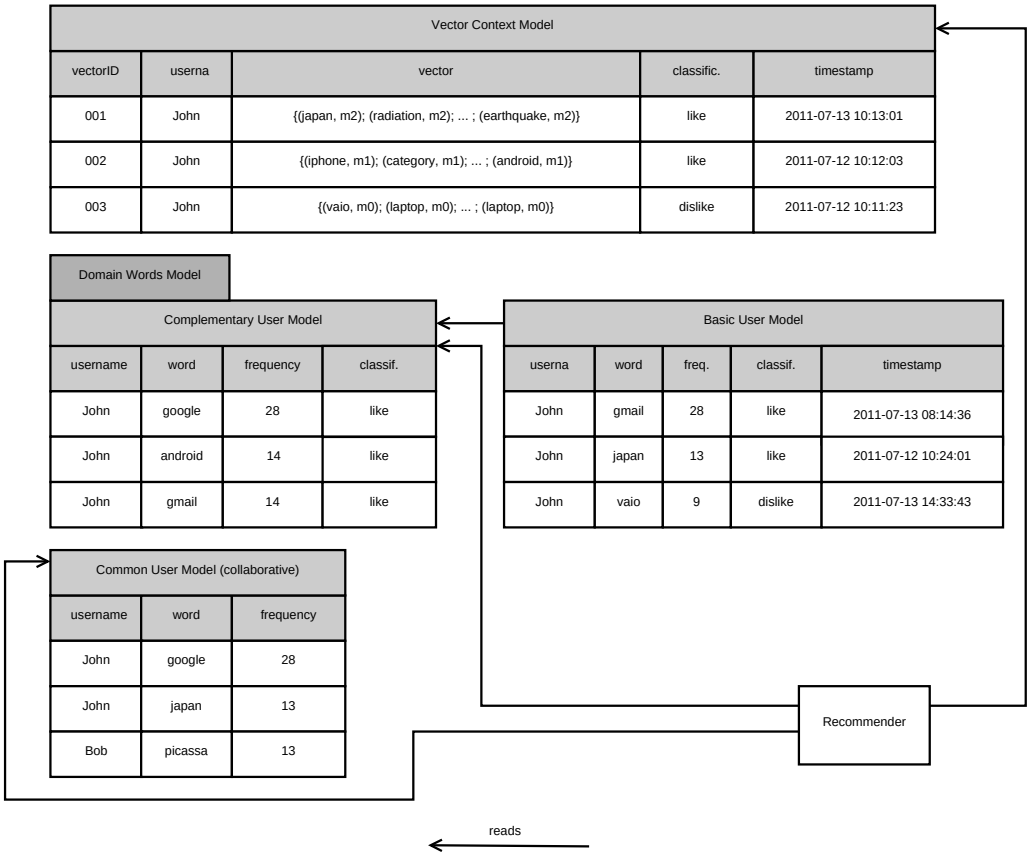

FIGURE 3.2: Extended User Model for user interests representation.

3.2.2 Complementary User Model

This user model is the product of the information provided by the Basic User Model and the interpretation done by the Profile Creator according to the Domain Words Model. This model contains the top n words from the Basic User Model and the related words from the Domain Words Model that have the correct association.

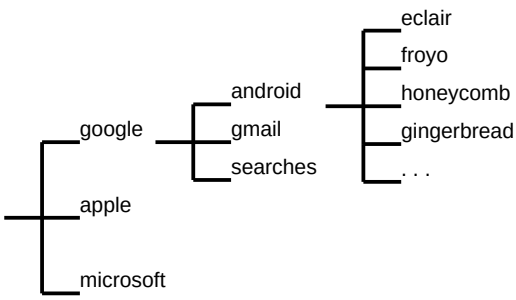

FIGURE 3.3: Representation of the Domain Words Model.

The Complementary User Model can be represented as $U_i = ((w_1, f_z), (w_2, f_y), ..., (w_n, f_a))$ for both, liked and disliked messages, where w is a word that is either in the Domain Words Model or a word that belongs to a set of classified messages by a user i with

frequency f in descending order where $z > y$ and n is the average number of words of every classified message without considering the repeated words.

3.2.3 Context Vector Model

The Context Vector Model is a set of vectors that allows the message context representation of what has been classified by the user. These vectors permit the Recommender to interpret the relationship among words and suggest the best message. The attributes of the vectors are necessary to identify the ownership of a vector, what the classification given by the user is, and the date when the classification has been done. This last attribute is explained in the subsection 3.3.3 *Adaptation*. Figure 3.4 shows a representation of a vector regarding the user and a message *m1*.

002	John	{(iphone, m1); (category, m1); ...; (android, m1)}	like	2011-07-12 10:12:03

FIGURE 3.4: Vector representation for context evaluation of the message *m1*.

3.2.4 Common User Model

The Common User Model is the product of the collaborative filtering algorithm that allows the representation of interests of a group of users with the same kind of interests. This model also provides complementary information to their respective Extended User Models. This partial model is not persistently stored due to the fact that the users similarities, during the evaluation of the Basic User Models, are constantly changing. In this concept, the common user interests are represented by a list of words with their corresponding frequencies. Figure 3.5 shows how the Common User Model is represented according to the Basic User Model between two users. In order to evaluate the context of the users' interests, the same Context Vector Model is used. The definition and usability is explained in section 3.3.4.2 *Common User Model*.

In this concept it has been identified that the Common User Model may represent the interests of various users through two main approaches:

- **Cluster representation:** A common user model is created according to a cluster of users with the same preferences. The following picture shows two different clusters with their corresponding users and Common User Models.

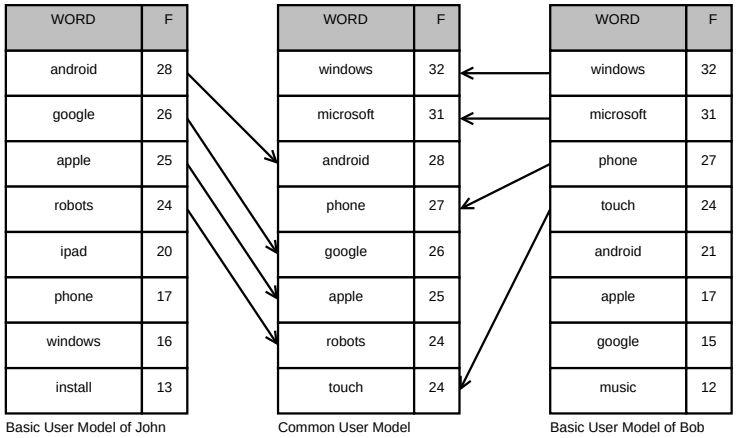

FIGURE 3.5: User model representation between two users with common interests.

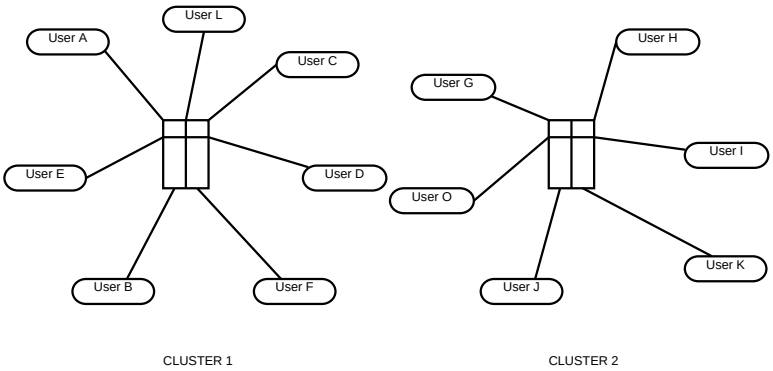

FIGURE 3.6: Cluster representation of Common User Model.

This approach, regarding the memory cost of the analysis of the users' preferences, is more efficient. However, the recommendations may not be as accurate as expected since the Common User Model may contain various terms from other users that may not be interesting to a specific user.

- **Binary representation:** A common user model is created between two users with same preferences. Figure 3.7 shows a binary representation of the Common User Model between two users.

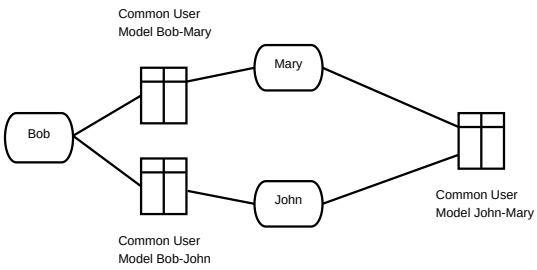

FIGURE 3.7: Binary representation of a Common User Model.

This approach guarantees better recommendations than the one provided by the Cluster representation approach, since the most important terms are considered in the Common User Model. The disadvantage of this approach is its higher memory consumption cost because one Common User Model is created for every pair of users.

3.3 Extended User Model Generation

This section explains a series of phases evaluated by the Profile Creator in order to generate an Extended User Model for each user according to his or her classified messages.

3.3.1 Phase 1: Creation of the Complementary User Model

In this phase, the Complementary User Model is created from the Basic User Model. The aim of this phase is to create a first instance of the user model with already user-classified words that are explicitly related to the messages. Figure 3.8 shows a flow diagram of the first phase. In order to complete this phase, the Profile Creator evaluates the following aspects:

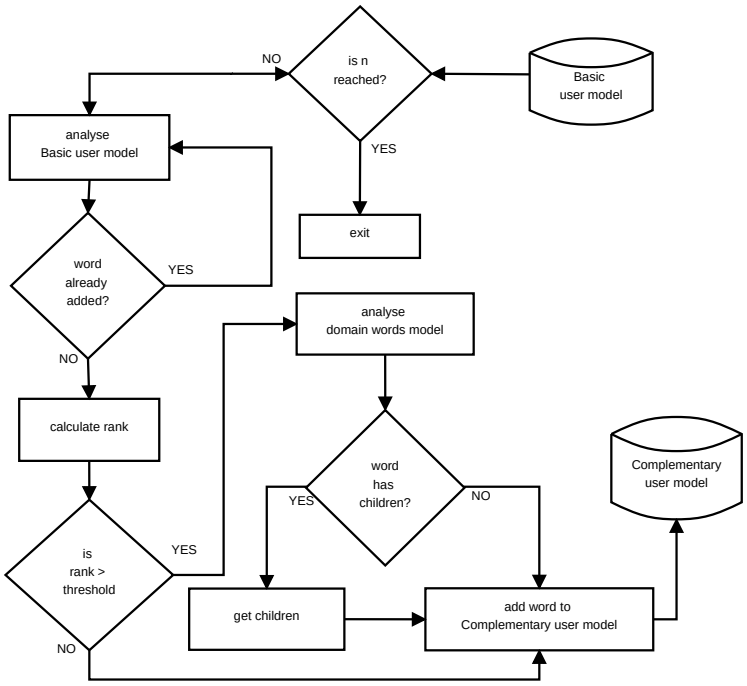

FIGURE 3.8: Diagram flow of the phase "creation of the Complementary User Model".

3.3.1.1 Analysis of the Basic User Model

In this procedure, the Profile Creator evaluates the Basic User Model by analysing the top n words sorted by their frequency in descending order[3]. For each word evaluated, the Profile Creator analyses whether the word has already been added into the Complementary User Model or not. If the word has been added, then the Profile Creator evaluates the next word until the value n is reached. However, if the word has not been added, the Profile Creator will calculate its rank.

3.3.1.2 Rank Calculation

The rank has been created in order to understand how important a word is according to the bank of words that already exist in the user model. This rank allows the Profile Creator to evaluate if a word is important enough to find related words in the Domain Words Model. If the word's rank is of a higher importance, then the Profile Creator will add it to the Complementary User Model. The word frequency by itself is not a good determinant to evaluate the importance of a word because is necessary to have a point of reference to evaluate whether a word is important or not. To determine a reference, the maximum word frequency is used.

The rank evaluates the importance of a word according to a given value between 0 and 1. Every word with a rank higher than this value is considered important; this allows to search related words in the Domain Words Model. The rank of a word is directly proportional according to the maximum word frequency from the user model U_i. The following formula calculates the rank of a word according to the maximum word frequency and the word frequency:

$$rank = \frac{W_f}{F}$$

where W_f is the frequency of a word, and F is the maximum word frequency from the user model U_i. However, the problem with this representation is that if the maximum frequency is too high according to the second highest word frequency, the rest of the words will not be considered to be evaluated in the Domain Words Model. Therefore, the equation has been modified by:

$$rank = \frac{\log(W_f)}{\log(F)}$$

[3]Where n has been already defined in section 3.2.2 *Complementary User Model*.

The logarithmic function decreases the value of the frequencies when they are too high according to the average word frequency.

Once the rank has been calculated, the Profile Creator evaluates whether the word rank is greater than a certain threshold or not. This threshold is a value between 0 and 1 and, if this condition is true, then the word is considered important and therefore, related words that are in the Domain Words Model are added into the Complementary User Model. The frequency of the related words will not be the same as the evaluated word that has been selected from the Basic User Model, then, their frequencies are the frequencies of the main word divided by a coefficient c. For example, let us consider a coefficient $c = 2$. If a concept has a frequency $f = 27$, the related words of that concept will have a frequency $f = 14$. With this operation, the related word will not be considered important as the main word presented in the Basic User Model, instead, it will be considered as regular word.

3.3.1.3 Domain Words Model Analysis

In this step the Profile Creator searches for related words in the Domain Words Model to later store them in the Complementary User Model. In the hierarchy of the Domain Words Model, the related words of a concept are the direct children of it. However, not every concept may have immediate children; this situation occurs when the concept is already deep enough in the hierarchy to have leaf nodes. The pseudo-code that evaluates and adds the concept, and the related words to the Complementary User Model, is the following:

```
wordRank ← getRank(word)
if wordAlreadyAdded then
  checkBasicUserModel
else
  if wordRank ≥ coefficient then
    insert(word, wordRank)
    children ← getChildren(word)
    for all children do
      insert(child, wordRank)
    end for
  else
    insert(word, wordRank)
  end if
end if
```

Figure 3.3 shows an example of a part of a Domain Words Model. The algorithm represents the last steps of figure 3.8, where the word is analysed in the Domain Words Model until it is persistently stored for future updates. According to these updates or evaluations, the Extended User Model goes through the next phases in order to either learn or adapt to the new users' interests.

3.3.2 Phase 2: Correlation Evaluation

At this phase, the Profile Creator evaluates the context of a classified message, i.e. to understand why a user has *liked* or *disliked* a message. The context classification is relevant when two or more concepts, that are not appealing to a user, become interesting for him or her when those concepts together have a different sense in the same message. For example, a user may neither be interested in the word *earthquake* nor *Japan*. However, if both words are together in the same message such as *Big earthquake in Japan*, the user can classify the message but the frequency of both concepts may not be high enough to be added into the Complementary User Model.

In order to analyse the context of a classified message, it is important to associate all the words from the message in a list of concepts. In this list of words, the word frequency does not affect the creation of the user model, because the list is considered as a vector of words that will be compared with unclassified vector messages. For vector comparison, the Cosine Similarity is used and the top n similar messages are displayed, where n is an arbitrary number set by the user.

As presented in section 3.2.3 *Context Vector Model*, the Context Vector is a model that each classified message has. The vector consists in a set of keywords and a correlation of words that belong to the classified message. These important words are added to the vector once the message has gone through the pre-processing phase. Every user has a stack of Context vectors with their corresponding timestamp to evaluate whether the Context vector is valid or not. A representation of the Context vector can be seen in figure 3.4.

3.3.3 Phase 3: Adaptation

In this phase, the Profile Creator evaluates the validity of the Extended User Model regarding to the change of the users' preferences. An example of this requirement could be represented when a user has an interest on a certain topic, but after Δt, the user has changed his or her interests. So far, the importance of a set of words has been

established by their frequency, however, if Δt is short according to the user's needs, the system may not have enough time to increase the words frequencies. Although the frequency is a good determinant to evaluate whether a word is important or not, it does not determine a long term change of preferences. According to this statement, two terms can be defined: short-term preferences and long-term preferences.

3.3.3.1 Short-Term Preferences

In this period of time, the preferences are evaluated by the Profile Creator according to the word frequency and the top n words of the Basic User Model[4]. If a new concept is added to the Extended User Model, it will be evaluated according to its frequency until t is reached. In this period, the word frequency only increases. However, if the word does not increase enough its frequency (i.e. other words are increasing their frequency rapidly) it will not be considered within the top n words to be used to create the Complementary User Model.

The frequency is read by the Recommender to interpret the importance of a word and to evaluate whether new messages containing this word are important or not.

3.3.3.2 Long-Term Preferences

In this period, the preferences are evaluated by the Profile Creator according to the word's timestamp. Once that a long period t is reached, the frequency of the words, that have been already added in the Extended User Model, decreases in order to minimize the importance of a word. This aspect does not affect the recommendations if users do not change their preferences and keep classifying messages with the same terms that they already like. In order to decrease the word frequency, after this period, the frequency of each word is multiplied by a coefficient between 0 and 1; the numbers represents a percentage set by the user to decrease the word frequency (slower or faster). If the word frequency is less than 1, then it must be removed from the Basic User Model.

The adaptation mechanism is also used in the context analysis as an attribute of the generated vector. This analysis is performed at the very moment the user classifies a message. This adaptation attribute evaluates whether the vector still represents the users' interest or not. After a Δt the vector will be deleted and its place in the stack will be taken by the next vector that comes after the deleted vector. Figure 3.9 shows how

[4]Where n has been already defined in the subsection 3.2.2 *Complementary User Model*.

the timestamp attribute changes after t is reached and figure 3.10 shows an example of how the timestamp is used in the Context Vector Model.

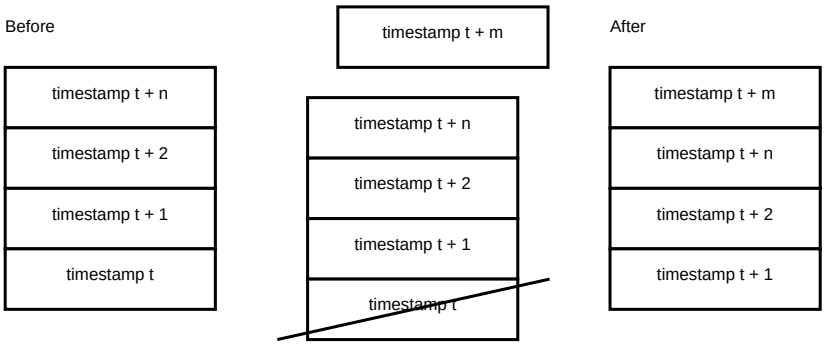

FIGURE 3.9: Timestamp representation regarding lifetime and positioning.

Before

vectorID	username	{(word A, mn); (word B, mn); ... ; (word N, mn)}	classification	timestamp t+n
vectorID	username	{(word A, m2); (word B, m2); ... ; (word N, m2)}	classification	timestamp t+2
vectorID	username	{(word A, m1); (word B, m1); ... ; (word N, m1)}	classification	timestamp t+1
vectorID	username	{(word A, m0); (word B, m0); ... ; (word N, m0)}	classification	timestamp t

After

vectorID	username	{(word A, mn); (word B, mn); ... ; (word N, mn)}	classification	timestamp t+m
vectorID	username	{(word A, m2); (word B, m2); ... ; (word N, m2)}	classification	timestamp t+n
vectorID	username	{(word A, m1); (word B, m1); ... ; (word N, m1)}	classification	timestamp t+2
vectorID	username	{(word A, m0); (word B, m0); ... ; (word N, m0)}	classification	timestamp t+1

FIGURE 3.10: Example of timestamp applied to Context Vector Model.

3.3.4 Phase 4: Collaborative Filtering

In this phase, a different aspect of collaborative filtering is applied in the Extended User Model. The common aspect recommends the same item to a user who has similar preferences as another user. However, the same item cannot be recommended if two users have similar Basic User Models but different sources of information (URL feeds). In this subsection, the steps, that the Profile Creator performs until the collaborative filtering is completed, are explained.

3.3.4.1 Cosine Similarity Evaluation

Two or more users have the same preferences if the Cosine Similarity between them is higher than a certain value. This value is calculated by using the top n words, ordered by their frequency, from the Basic User Model of every user; where n is the average number of words of the messages that the user has already selected. If the similarity between two users is high enough according to a coefficient, a temporary Common User Model is created.

3.3.4.2 Common User Model

Once that the Cosine Similarity between two users has been evaluated and fulfilled the condition mentioned above, a temporal Common User Model is created in order to be evaluated by the Recommender. The Common User Model is now used to suggest the users recommendations according to their own unclassified messages. The user model is temporal because user similarities are changing constantly during both, short-term preferences and long-term preferences. The Common User Model is created by using the top $n/2$ words, sorted by frequency, from the Basic User Model of every user. At the same time, the Profile Creator will not add words that another user already has, and it will retain the word with its frequency. Figure 3.4 shows an instance of the Common User Model AB between the Basic User Models from user A and user B.

With the creation of the Common User Model between two users, the Profile Creator concludes the instantiation of the Extended User Model, and it stores the model persistently in order to be analysed by the Recommender. The next section describes the details of how the Recommender interprets the Extender User Model to recommend unclassified messages.

3.4 Recommendations

This section presents how the Recommender performs, interprets and suggests the unclassified messages from a user according to his or her Extended User Model.

Once the Profile Creator evaluates all the presented phases above, the Recommender analyses the Extended User Model and decides whether the Cosine Similarity or the

Naive Bayes Classifier[5] must be used to calculate the recommendation. The Recommender evaluates the Complementary User Model, the Context vector as a non-collaborative filtering, the Common User Model and the Context vector for collaborative filtering. Each step is explained in the following subsections.

3.4.1 Recommendations over Complementary User Model

The recommendations performed over the Complementary User Model are calculated by using the Naive Bayes Classifier. This classifier takes into account the words that have been classified as *like*, *dislike* and the frequency of the classified words. The higher the frequency of a word, the higher the probability of a message to be selected if it contains that word.

Once the Recommender has evaluated the classified messages, it calculates the probability of the message to be selected as recommended. In this case, two options can be displayed to the user: all the messages that have a probability greater than a value P, or the top n messages with higher probabilities. In any case, the unclassified messages will be marked as "recommended by Complementary User Model"

3.4.2 Recommendations over Context Vector

This recommendation evaluates the context of the classified messages by using the Cosine Similarity. This similarity value is used because the context of a message is represented as a vector of words, and it is compared with the incoming messages that have not been classified by the user. In this step, the incoming message is transformed into a vector and compared with every vector that the user currently has (according to their timestamp value in which older vectors are dropped). If the new message converted to a vector is similar to a vector that the user has, then the message is classified as "recommended by Context vector".

3.4.3 Recommendations According Collaborative Filtering

In the collaborative filtering recommendation, two different evaluations are performed: evaluation of the Common User Model and evaluation of the Context vectors. In the first evaluation, the Recommender analyses the Common User Model that has been created by using the similarities over two users. At the same time, it evaluates the unclassified

[5]Explained in chapter 2 *Background and State of the Art.*

messages of both users. The recommendation is done in the same way as the Complementary User Model is analysed, i.e. with the Naive Bayes Classifier. Figure 3.2 shows how the Common User Model retains the word frequency in order to be classified by this method. If the messages satisfy the probability condition, it is marked as "recommended by collaborative filtering - Common User Model"

Regarding the context recommendations of the collaborative filtering, the same Context vectors of each user are used but evaluated in a different way. When two users have the same kind of interests, a determinant $\bar{I}$ is calculated as follows:

$$\bar{I} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} Cos(A_i, B_j)}{n+m}$$

where n and m are the number of vectors of user A and B. Once that $\bar{I}$ is calculated, every vector from user B is used to evaluate the unclassified messages from user A. Once again, the Cosine Similarity function is used to evaluate every message and for every operation whose result is greater than $\bar{I}$, the message is classified as "recommended by Collaborative filtering - context".

3.4.4 Combined Final Recommendation

This is the last step performed by the Recommender. Once the Recommender has evaluated the recommendations regarding the Complementary User Model, Context Vector Model and the Collaborative filtering, the Recommender decides whether the message is finally recommendable or not.

It is very important to mention that the recommendations will not always be accurate. Nevertheless, this accuracy can be set in different levels. For example, low accuracy, where the message is recommended if one of presented recommendations satisfy its conditions. Medium accuracy, where the message is recommended if two of the presented recommendations satisfy the conditions. Finally high accuracy, where the message is recommended if all of the presented recommendations satisfy the conditions. Figure 3.11 represents the different options the user has in order to see the recommendations: fewer recommendations, more accuracy. More recommendations, less accuracy.

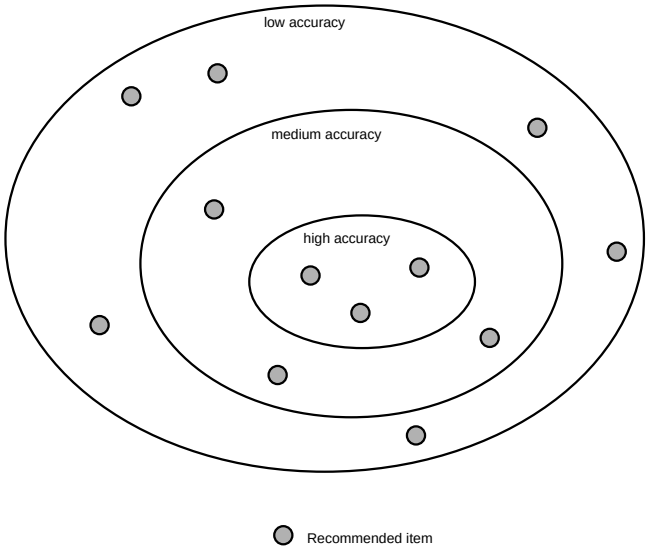

FIGURE 3.11: Recommendations according to their accuracies.

3.5 Summary

This chapter presented the concepts of the contribution of this thesis. The first section explained how the components of the recommender system architecture, such as the Profile Creator and the Recommender, interact with the user models. The second section introduced the Extended User Model, which represents the users' interests in various ways. These ways are represented with the Complementary User Model, Context User Model and Common User Model. Finally, how the Recommender suggests messages using the Cosine Similarity and the Naive Bayes Classifier, was explained. Table 3.1 describes each user model to have a quick reference.

Basic User Model	Represents the user interests by counting the number of words that appear in *liked* and *disliked* messages.
Complementary User Model	Takes a number of *liked* and *disliked* words in order to find associated words and, therefore, provide alternative recommendations.
Context User Model	Stores *liked* articles as a vector to identify and represent the context of new messages.
Common User Model	Represents the users' interests with similar preferences by taking the *liked* and *disliked* words from their according Basic User Model.
Extended User Model	Analyses the results of each user model to combine their results and provide more complete recommendations.

TABLE 3.1: Summary of the presented user models.

Chapter 4

Implementation

In this chapter the details of the implementation of the recommender system based on both, Basic User Model and Extended User Model are explained. First, the selected technologies such as programming language and databases are presented. After that, some aspects of the architecture, which interact with the recommender system, are described. In the third part, the core engine package system is detailed. Next, the implementation of the Extended User Model is introduced. In addition, the implementation of the Recommender follows this section. Finally, a summary of the entire chapter is outlined.

4.1 Development Languages

Java and PHP, both used in Eclipse IDE, were the two programming languages selected to implement this concept previously presented in chapter 3 *Description of the Concept*. Besides Java and PHP, SQL was widely used to process and evaluate important information attached to the user models tables. Nevertheless, SQL is not considered a programming language itself, but a structured query language. PHP was selected because it is fast, stable, secure and provides connectivity to MySQL, which is necessary for this work due to its multiples connections to the database. The engine or core, that creates and calculates the recommendations, were implemented in Java; a programming language that seems more suitable due to all its methods already implemented, such as Cosine Similarity and Naive Bayes Classifier, as opposed to other similar programming languages. Finally, a MySQL database stores the user models and messages. The MySQL database is the mediator between both programming languages.

4.2 Selected Aspects of the Implementation

This section explains some aspects of the implementation of the graphical user interface and the database modelling. These aspects of the recommender system are implicitly connected to the Extended User Model and the Recommender; this means that they could be implemented in a different fashion.

4.2.1 Graphical User Interface

The graphical user interface is a web page that was implemented in PHP. In this web page, in order to download the messages, the user must provide the URL feed, which is downloaded by another PHP instance. This instance is executed once the user is on the welcome screen of the website. In this implementation, for evaluations purposes, the URL feed from a technology blog named "Engadget"[1] was used.

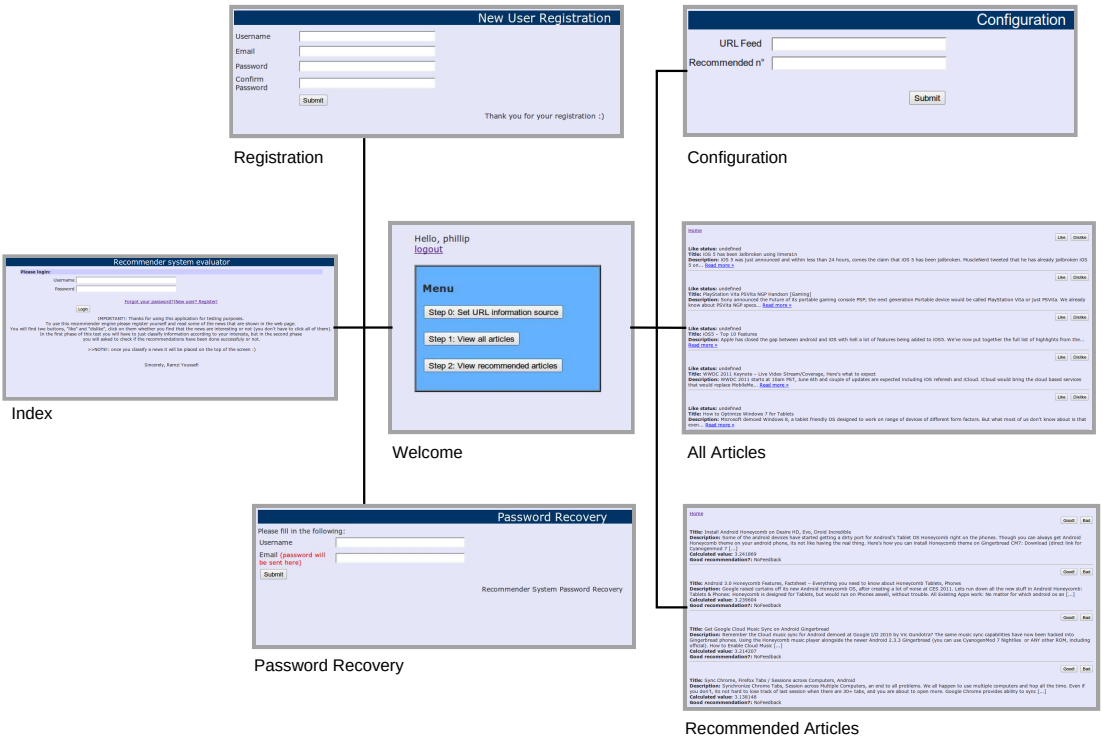

FIGURE 4.1: Graphical user interface navigation.

Figure 4.2 shows the navigation options that the user has. In the first instance, the index page is loaded where the user has the possibility to either register, login or recover his password. In the welcome screen, the user has the possibility to select three different alternatives: configuration, display feeds and display recommended feeds. In the configuration option, the user can set some parameters regarding the recommender system, for

[1] http://www.engadget.com - Last visit July 2011.

example, to set the number of recommendations displayed on the screen. At the same time, the configuration option allows the user to set the URL in order to download the messages. Messages are displayed on the "all feed messages" windows, every message has the option *like* or *dislike*, which allows the user to indicate his or her interests. In addition to this capability, the user has the option to read, in the "recommended feed messages" page, recommended articles that have been calculated by the recommender system according to what he or she has chosen previously. This recommended message page also allows the user to give feedback of every recommended message to the Recommender by clicking on the buttons *good* or *bad*. A *good* feedback is sent if the presented message is correctly recommended or a *bad* feedback is sent if the presented message has not been correctly recommended.

4.2.2 Database Model

The database, which persistently stores the downloaded messages, has been realized using MySQL. As explained in the previous section, PHP is used to get the feeds from a given URL in order to store the messages. These messages are stored in a MySQL table called **InformationExplosion**, which contains the following information:

- **dataID:** Allows the message to have a single identification. DataID is auto incremental which ensures uniqueness; it prevents the messages from having the same URL message in the case that different users have input the same URL feed.
- **urlMessage:** Stores the URL of the message. The stored information can be provided to the user in order to visit the original article.
- **titleFeed:** Stores the title of the article.
- **bodyMessage:** Stores the body of the article.
- **like:** Stores the classification of the message - *like* or *dislike*.
- **basicChecked:** Controls whether the message has been selected to construct the Basic User Model or not.
- **contextChecked:** Controls whether the message has been selected to construct the Context Vector Model or not.
- **userid:** Indicates whom the message belongs to.
- **commonRecommended:** Shows the maximum float[2] number that the message might have calculated according to the collaborative filtering recommendation.

[2]Since Cosine similarity and Naive Bayes classifier return float values, these are stored to analyse and represent recommendations in different ways.

- **complementaryRecommended:** Shows the maximum float number that the message might have calculated according to the Complementary User Model recommendation.
- **contextRecommended:** Shows the maximum float number that the message might have calculated according to the Context User Model recommendation.
- **basicRecommended:** Shows the maximum float number that the message might have calculated according to the Basic User Model recommendation.
- **Feedback:** Controls, whether the recommended message, evaluated according to the Extended User Model, has been done successfully or not.

The Basic User Model is stored in a table called **BasicUserModel** with the following fields:

- **idAutoInc:** Identifies an inserted word with an auto incremental number.
- **userid:** Identifies whom the words belong to.
- **word:** Stores a single word.
- **frequency:** Stores the frequency of the word.
- **likeness:** Shows whether the word is liked or disliked
- **taken:** Controls whether the word has been analysed to build the Complementary User Model or not.
- **timestamp:** Controls when the word is added or updated.

The Complementary User Model is stored in a table called **ComplementaryUserModel**, and it has almost the same fields as the BasicUserModel table. The fields are **idAutInc, user, word, frequency** and **likeness**. To represent the context of the interests with a vector, a table named **ContextVectorModel** was created with the fields **vectorID, username, vectorString** and **timestamp**. The vectorString field stores the most significant words of a message.

Finally, a table named **User** that stores users' information was created with the following fields:

- **userid:** Stores the users' ID.
- **uname:** It shows the username that has been chosen by the user.

- **password:** It contains the encrypted password of the user.
- **email:** Stores the e-mail of the registered user.
- **dateJoined:** It shows when the user has been registered into the system.

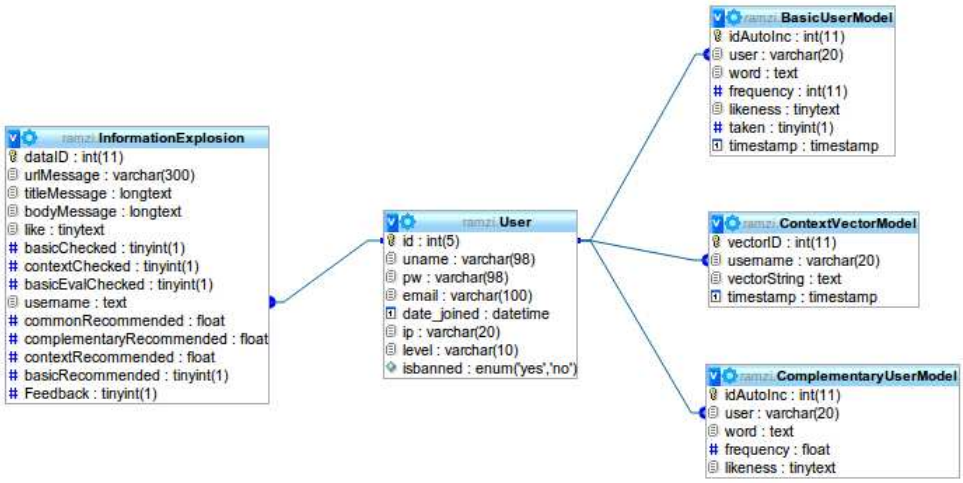

FIGURE 4.2: Relational Model of the recommender system.

To summarize what has been described above, the figure 4.2 illustrates the relational model of the database.

4.3 Packages Diagram

This section presents how the core of the recommender system was implemented. To understand how the implementation is structured, the Figure 4.2 illustrates the package diagram of this work.

As shown in the figure, the core component of the recommender system was implemented with three packages that are explained in detail in the following subsections.

4.3.1 Profilecreator Package

As described in chapter 3, *Description of the Concept*, this package contains the necessary sub-packages and classes that allows the Profile Creator instantiate the presented user models. A detailed description of each package that belongs to the Profilecreator is presented as follows:

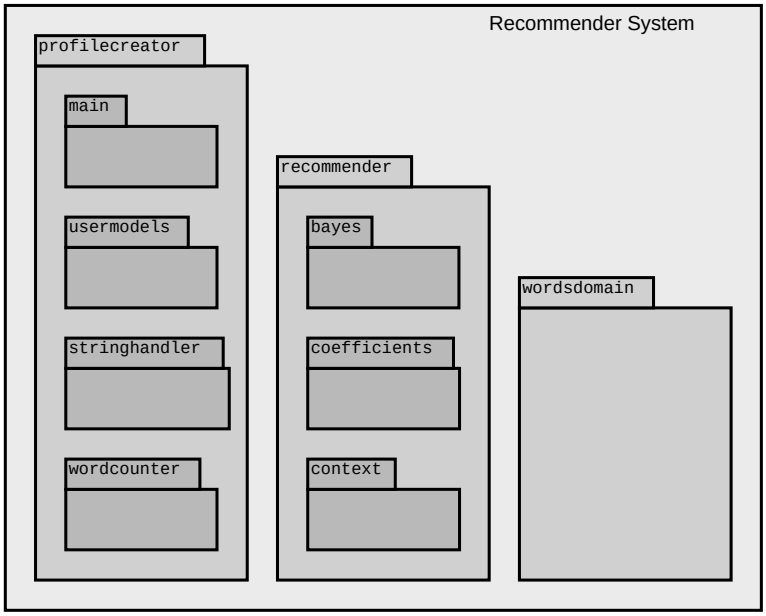

FIGURE 4.3: Java package diagram.

- **Usermodels:** This package contains the user model definitions for the Basic User Model, Complementary User Model, Common User Model and the Context User Model. These definitions are Java classes that contain the necessary methods to instantiate the user models. However, in order to create the user model, these methods invoke classes and method from other packages.

- **Stringhandler:** This package contains the necessary classes to convert, clean and represent the strings of the messages. In here, the class Tokenization converts an string into a list of separated words in order to analyse the information in a better way. This package also contains the data type that represents a word and its frequency.

- **Wordcounter:** This package contains two classes: IntermediateValue and WordCounter. The IntermediateValue is a data type that represents the word with its frequency and also the user who owns the word. The WordCounter counts the number of times that a word is present in a message.

- **Main:** This is the package that starts the recommender system. The class MainBasicUserModelCreator instantiates the Basic User Model and the Recommender. The MainExtendedUserModel instantiates the Complementary User Model, the Context User Model and the Common User Model with their according recommendations. The class MainResetRecommender[3] is also presented in this package,

[3]This class has been created for evaluations purposes and it is not necessary if the recommender system is running normally.

which reset and deletes all the information of the database except the downloaded messages, their classification and the users of the system.

4.3.2 Recommender Package

This package contains the packages and classes that evaluate the user models in order to do the recommendations. A detailed description of each package is presented as follows:

- **Bayes:** This package contains the necessary classes to perform recommendations using the Naive Bayes Classifier. The class SpamFilter contains the methods to calculate the probabilities but not to perform the recommendations. To do this task, the BayesRecommender class performs the recommendations regarding the user models, once the probability has been calculated by the SpamFilter. Other classes are used as a data type to represent a word with its frequency for both, liked and disliked messages.
- **Coefficients:** This package has the classes of the coefficient similarities such as Cosine, Dice and Jaccard. However, only the Cosine Similarity is used in this work. The CosineSimilarity class contains the algorithm to calculate the similarity between two lists of words or two strings.
- **Context:** This package contains a single class called ContextRecommender. This class reads the vectors from the database and evaluates the recommendations with the Cosine Similarity.

4.3.3 Wordsdomain Package

This package contains the Domain Words Model, a set of words with special relationship between them. It was implemented in Java using Jtree - Swing, an API for providing graphical user interfaces for Java programs. In this implementation, the created Jtree contains terms related to the topic "information technologies" which is fully related to the technology blog used in this implementation. The figure 3.3 shows an example of the implemented Domain Words Model. The package also contains a class called StopWords that includes a List of non-important words in order to be instantiated whenever the Profile Creator requires it.

4.4 Extended User Model Instantiation

This section explains how the Extended User Model was implemented. The implementation of this concept is presented according to the steps described in the chapter 3, *Description of the Concept*, in the section 3.3 *Extended User Model Generation.*

4.4.1 Creation of the Basic User Model

In order to create an Extended User Model, a Basic User Model must be utilized. The Basic User Model was implemented by retrieving the list of users that were registered in the recommender system. For each message, that has been classified by a user with the option *like* or *dislike*, the Profile Creator stores the information in a list with the presented data type *(username, word, value).*

Once that a classified message has been analysed, the field "basicChecked" of the message changes to "YES". This field controls that the message is not analysed again; because the Profile Creator evaluates this field periodically. This procedure is done for both, like and disliked messages. The returned list is analysed to remove repeated words and to store the list in the MySQL table BasicUserModel. Once this user model is completed for all users, the Complementary User Model is created.

4.4.2 Creation of the Complementary User Model

The first evaluation that the Profile Creator does, according to the Complementary User Model instantiation, is to get the maximum word frequency of the Basic User Model of one user. This evaluation is not written in Java, but in SQL due to its simplicity and rapidity. For example, the function MAX() is used to get the maximum frequency of a list of words from a user.

Once the maximum word frequency is calculated, the top n words from the Basic User Model[4] are used. An SQL query is used to get the "top" n words, in order to carry out this procedure, the function ORDER BY DESC LIMIT 0, n is used. The Profile Creator updates the field "taken" of every word that has been used to create the Complementary User Model. This update changes the field "taken" by a "YES" status.

In this phase, the rank is calculated according to the formula presented in the section 3.3.1.2, *Rank Calculation.* In this implementation, the rank is calculated with the

[4]The number n is explained in the section 3.2.2, *Complementary User Model.*

following Java code:

```
float rank = (float) ((Math.log10(freq))/Math.log10(maxFrequency + 1));
```

Once the rank is evaluated, the algorithm presented in the section 3.3.1.3, *Domain Words Model Analysis*, is executed to create the Extended User Model.

4.4.3 Creation of the Context Vector Model

In this phase, a Java class was created to represent a context vector as a data type (vectorID, username, vectorList) where **urlFeed**, **titleFeed** and **bodyFeed** are added together to represent them as one vector in the vectorList variable. It is important to consider that all this vector handling implies string cleaning and word removing of stop words. For every message that has been analysed, the field contextChecked of the BasicUserModel table is updated to "YES", this action is done to avoid a double-check of the messages by the Profile Creator while the Context Vector Model is instantiated. Once this aspect of the message is evaluated, the contexts are stored in the table ContextVectorModel.

4.4.4 Collaborative Filtering Analysis

To perform a collaborative filtering, the Profile Creator evaluates the Cosine Similarity between two users by retrieving the first n words[5] of the Basic User Model of each user. In this implementation, and because of the similarities found between users in the system evaluations, a value of 0.4 was set as a determinant to see if two users have the same kind of interests. Once that the evaluation is performed, a temporal Common User Model is created.

To create a Common User Model between two users, the *n/2* amount of words of each Basic User Model must be retrieved and merge them. This total amount of words completes an entire user model. This procedure is done for both, liked and disliked messages because these two branches are the inputs of the Naive Bayes Classifier. The figure 3.5, presented in the chapter 3, illustrates how the Common User Model is represented.

[5]Where n is the average number of words of every classified message without considering the repeated words. Explained in section 3.2.2 *Complementary User Model.*

4.4.5 Validity Control According to Timestamp

The timestamp field, in the presented user models, was implemented with the option given by MySQL when the table is created. To control whether the timestamp is old or not, regarding the user change of interest, an SQL trigger is set:

```
CREATE TRIGGER oldTimestampDelete
AFTER INSERT ON BasicUserModel
FOR EACH ROW
BEGIN
DELETE FROM ContextVectorModel
WHERE timestamp < (NOW() - INTERVAL 10 MINUTE);
END
```

This trigger allows the Extended User Model to have the current Context vectors in order to be used by the Recommender. The presented trigger, instead of deleting a Context vector, could decrease the frequency and minimize the importance of a word that is listed in the Basic User Model. This action is executed to avoid the word being selected by the Profile Creator when the Complementary User Model is created. For evaluations purposes, in this implementation, the time is set to 10 minutes.

4.5 Recommendation Analysis

This section presents how the recommendations are performed once the Extended User Model is created. The Recommender evaluates whether an unclassified message is recommendable or not. To do this, two approaches are necessary, the Cosine Similarity and the Naive Bayes Classifier. These approaches were implemented as follows:

- **Cosine Similarity:** This similarity measurement was implemented using an open source package[6] and it calculates the similarity between two strings considering each string as a vector of words. The output of this method is a float number between 0 and 1.
- **Naive Bayes Classifier:** The classifier has been implemented by Daniel Shiffman[7] as a Bayesian spam filter example. In the original code, the inputs of this classifier are two plain text files; one that represents all the emails marked as spam

[6] http://staffwww.dcs.shef.ac.uk/people/sam.chapman@k-now.co.uk/stringmetrics.html - last visit May 2011.

[7] http://www.shiffman.net; daniel.shiffman@nyu.edu - last visit May 2011.

and another file that represents non-spam emails. In this implementation, these input files were modified to a list of words marked as *liked* and *disliked*. Once the filter receives these changed inputs, it analyses the word, its classification and its frequency. According to this evaluation, the classifier reads a new message and evaluates the probability to be recommended.

These both approaches are used in different ways along the entire process. The recommendations over the Extended User Model are calculated once the user models are stored persistently (except the Common User Model). For every evaluated user model, a field in the InformationExplosion table is updated. In case of the Common User Model, the recommendation of a message is done "on the fly" because this user model is not stored persistently in the database.

In order to evaluate whether new messages are recommendable or not, the figure 4.2 shows how the implementation of the Recommender was implemented according to each user model.

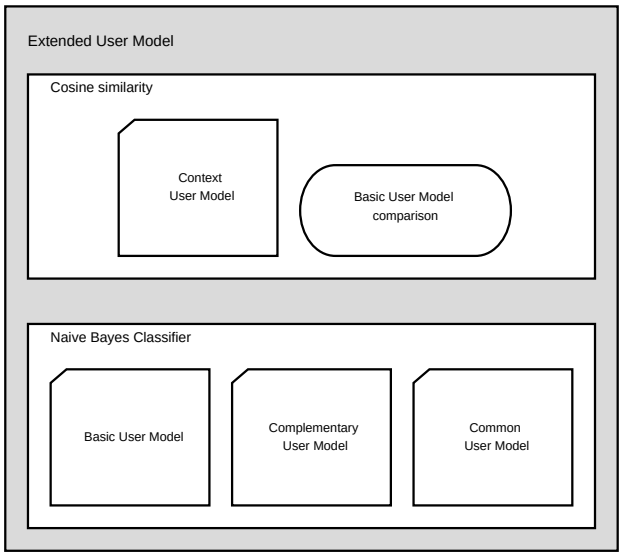

FIGURE 4.4: Cosine Similarity and Naive Bayes Classifier regarding to their user models and use.

This figure explains what is evaluated by the Cosine Similarity and the Naive Bayes Classifier. The figure Basic User Model Comparison, in the Cosine Similarity section, is represented by a different shape because it is not a user model but a process where the Cosine Similarity is used. In this process, the Cosine Similarity evaluates whether a Common User Model between two users must be created or not. Below, the implementation of this concept, regarding to a user model, is explained:

- **Cosine Similarity implementation:** The Cosine Similarity is implemented by retrieving the title and body of a message. These strings are joined together to later analyse them as one string. This method is used to compare two vectors, and at the same time, to evaluate the context of the message. An SQL query is used to get the top n words of the Basic User Model and calculate the user similarities:

  ```
  SELECT * FROM BasicUserModel
  WHERE user = ?
  AND likeness = ?
  ORDER BY frequency DESC LIMIT 0, ?;
  ```

  In this query, the LIMIT of words to be used has been established from 0 to 30, and *likeness* is a string with the value *like* or *dislike*. On this query, a list of words are concatenated as a string, as previously explained, to be used in the Cosine Similarity.

- **Naive Bayes Classifier implementation:** The Naive Bayes Classifier evaluates 3 phases: train *liked* words, train *disliked* words and finalize training. In both "training" phases, a similar SQL query, to the above presented, is used to get the words. In this training, the Naive Bayes Classifier evaluates the word, its classification and frequency to calculate the probability of the recommended message. For example, every new message that has a probability higher than 75 percent, is considered as recommended. However, in this implementation, every returned value is stored in the database. By doing this, the user can read the top messages with highest probability.

A different way of presenting the recommendations was implemented; this way consists of, storing the float (real) value of the evaluation in the database, and to have a better analysis and to avoid dropping data that might be necessary. In this work, a better value is stored when a better recommendation is found; for example, if the Cosine Similarity, between one context vector and a new message context vector, is 0.6, another context vector may have a similarity over the same new message of 0.63. In this case, the better value is stored. For example, $value = MAX(A_1; B_2; C_3; ...; X_n)$, where *A, B, C, ..., X* are similarity values between a new article and vector X_n, and n is the number of context vectors in the Context Vector Model.

To represent the top recommendations according to the Extended User Model, the following SQL query is used:

```
SELECT * FROM InformationExplosion
WHERE username = ?
ORDER BY contextRecommended + basicRecommended +
complementaryRecommended + commonRecommended
DESC LIMIT 0, ?;
```

On this query, the sum of each value is shown in a descending way. The limit values can be set by the user in the "configuration" web page of the presented graphical user interface.

4.6 Summary

The first section of this chapter explained that Java and PHP were the two programming languages used for the implementation of the recommender system due to their described advantages. The second section explained some aspects of the implementation such as the graphical user interface and the database model. The package diagram section showed and described the package structure used in this work. These three main packages represent the core of the recommender system. At the same time, these packages contain the necessary classes to perform the recommendations.

In order to explain how the Extended User Model is instantiated, this section described a set of phases. In addition, the database tables, Java code and SQL triggers were explained. This section described how the tables and user models interact, and how the fields of the tables change their status once the information is analysed. The first part of the last section presented how the Cosine Similarity and the Naive Bayes Classifier work. The second part, regarding the data structure of the recommender system, described how these evaluation measurements were implemented.

Chapter 5

Evaluation

This chapter presents a description and the results of the implemented recommender system's evaluation. First, a description of the evaluations is introduced to identify the parameters and the models that are compared. Second, an objective evaluation is performed according to a set of articles that have been already classified by users. The objective evaluation evaluates every single user model and the Extended User Model. Third, an evaluation according to users' feedback is presented. The real users' evaluation is a subjective evaluation that also considers every single user model and the Extended User Model. Finally, a summary of this chapter is presented.

5.1 Description of the Evaluation

In this chapter, two main evaluations were performed: an objective evaluation and a real user's evaluation. The first evaluation analyses articles that have been already classified to have more deterministic results. The second evaluation considers the user's evaluation to have more realistic results, because finally it is the user who decides whether a recommendation is good or not. The evaluations of this concept compare the presented user models, such as the Basic User Model and the Extended User Model. The evaluations consider also the models that build the Extended User Model: The Complementary User Model, that uses the Domain Words Model to add alternative words to the user model. The Context User Model, that evaluates the correlation among words to analyse the context of the user's preferences. Finally, the Common User Model, which is used as a collaborative filtering to build a user model using two users with same preferences. The user models described above are also evaluated separately to have a better analysis of the evaluation.

To compare and analyse each user model, some parameters or rules, that make the evaluation more fair, were set. For example, in the first evaluation, each user model has been created with the same percentage of articles. In other words, to evaluate the Complementary User Model with 30 percent of classified articles, the Basic User Model, the Complementary User Model and the Context User Model had to be also built with 30 percent of the classified articles. Another parameter is to set a threshold regarding the distribution results of each user model over the classified articles, which will be explained in the next section.

5.2 Objective Evaluation

The objective evaluation consist of the analysis of 100 different articles,[1] with around 200 words each article, classified by 7 different users. To create the user models, the amount of classified articles has been divided in two sets. The first set (SET0) contains the number of *liked* articles divided by two and the number of *disliked* articles divided by two. The second set (SET1) contains the rest of the *liked* and *disliked* articles; the items of SET1 must be evaluated according to each user model to check whether it can predict the articles' classification. This evaluation analyses if a larger number of classified articles implies better recommendations. To do so, 10 percent to 100 percent of *liked* and *disliked* articles of SET0 are analysed to evaluate the tendencies of each user model.

To evaluate the precision of the user models, according to a user, the formula $Precision = i/t$ was applied, where i are the articles from SET1 identified as *like* by the recommender and t are the total amount of articles classified as *like* that are in SET1. To calculate how precise are the recommendations regarding each user model, a threshold has been set to identify whether an article is recommendable or not. In this evaluation, after analysing the distribution results obtained by the user models, the following thresholds were set:

- Basic User Model: 0.7
- Complementary User Model: 0.7
- Context User Model: 0.04
- Common User Model: 0.5
- Extended User Model: 0.5

[1] http://www.engadget.com - Last visit July 2011.

These values were selected after analysing the distribution of the results of the classified articles regarding each user model. In order to determine a user model's threshold, the maximum and the minimum recommendations values[2] were considered. After that, the threshold was set around the 50 or 60 percent of the total distribution, considering also that the distribution of *liked* and *disliked* articles is fair regarding this value. In another words, the threshold should represent more *liked* classified articles above this value than *disliked* messages. The Extended User Model is a especial case since it does not provide a similarity value from 0 to 1 but a sum of each result of the user models above. To calculate the sum of the user models' values, the second SQL query from section 4.5 *Recommendation Evaluations*, is used. The maximum sum of the values provided by the query is calculated as 100 percent and each article with a value over 50 percent of the maximum sum is identified as recommended.

5.2.1 Evaluation of the Basic User Model

The Basic User Model evaluation consider the result of the procedures above described. Table 5.1 shows the average results of 7 different users according to the user model.

Basic User Model	**Precision**
Articles 10%	0.16
Articles 20%	0.26
Articles 30%	0.33
Articles 40%	0.35
Articles 50%	0.33
Articles 60%	0.31
Articles 70%	0.36
Articles 80%	0.31
Articles 90%	0.37
Articles 100%	0.38

TABLE 5.1: Evaluation results of the Basic User Model with the creation of the user model considering 10% to 100% of classified articles.

These results show that the Basic User Model, as long as the user model evaluates more articles, the precision increases until 40 percent of classified articles is used. After this point, its precision starts to decrease gradually. One possible reason for this behaviour is that, at certain point, the user model considers the same terms. This makes the system to recommend always the same articles. In addition, after considering 60 percent of classified articles, the Basic User Model starts to increase its precision but almost in a

[2]In this evaluation, the recommendation values were calculated by creating the user model with 10 percent of the classified articles.

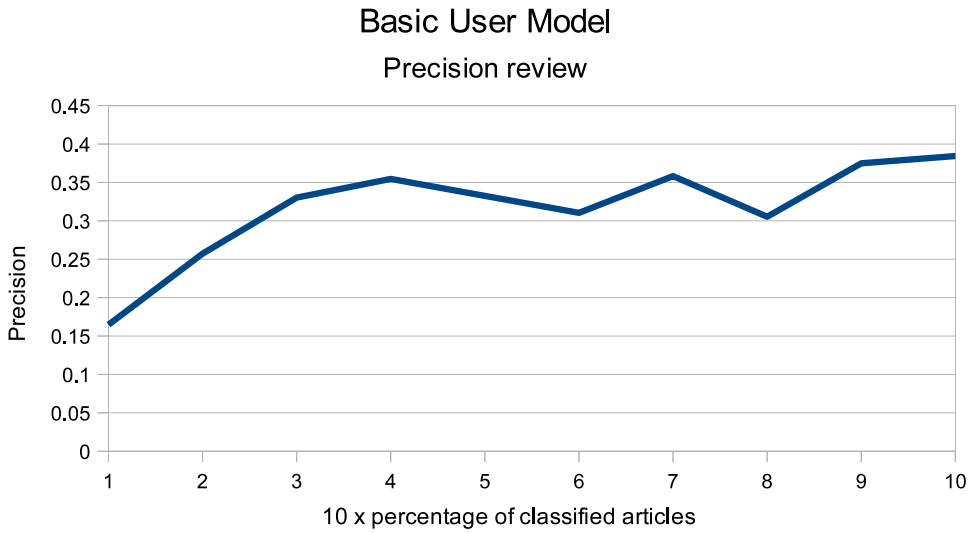

FIGURE 5.1: Data analysis of the recommendations according to the Basic User Model.

unpredictable way. Figure 5.1 shows a graphical result of the evaluation. The figure concludes that the learning process of the Basic User Model requires a big number of classified articles to give more precise recommendations. This conclusion is obtained after considering that the increase of precision between 40 percent and 100 percent of the classified articles is just 3 percent.

5.2.2 Evaluation of the Extended User Model

In this section, the models that build the Extended User Model are evaluated separately in order to have an overview and a point of comparison between the presented user models.

5.2.2.1 Complementary User Model

The Complementary User Model evaluation considers the result of the average values of 7 different users. Table 5.2 and figure 5.2 show the results.

The Complementary User Model increases its precision until 30 percent of classified articles are used to create the user model. After this percentage, the Complementary User Model gains stability until 80 percent of classified articles is used. From this point, the user model increase its precision almost 10 percent but in the end, its precision does not go beyond 35 percent. The results of this evaluation suggest that using the Domain Words Model makes the recommender more stable but less precise than the Basic User

Complementary User Model	Precision
Articles 10%	0.21
Articles 20%	0.26
Articles 30%	0.31
Articles 40%	0.31
Articles 50%	0.30
Articles 60%	0.29
Articles 70%	0.31
Articles 80%	0.27
Articles 90%	0.36
Articles 90%	0.35

TABLE 5.2: Evaluation results of the Complementary User Model with the creation of the user model considering 10% to 100% of classified articles.

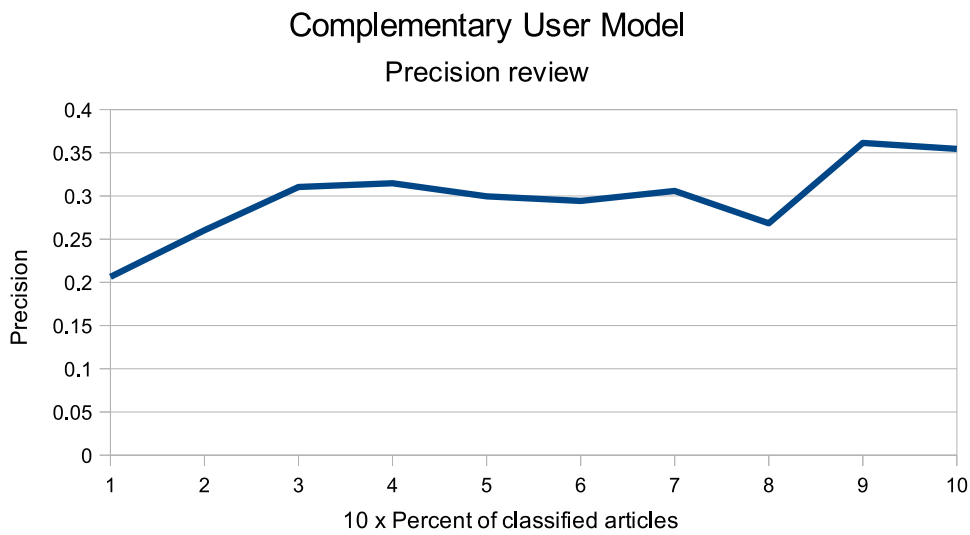

FIGURE 5.2: Data analysis of the recommendations according to the Complementary User Model.

Model. These low precision is obtained due to the Complementary User Model contains a number of related concepts that are not contained in what the user has classified. For the same reason, the Complementary User Model is not flexible enough to learn properly when more articles are used to build the model.

5.2.2.2 Context User Model

The Context User Model evaluation considers the result of the average values of 7 different users. Table 5.3 shows the obtained results.

According to table 5.3, recommendations regarding the Context User Model increase constantly. This is due to the Context User Model stores the context of the classified

Context User Model	Precision
Articles 10%	0.42
Articles 20%	0.59
Articles 30%	0.71
Articles 40%	0.76
Articles 50%	0.80
Articles 60%	0.84
Articles 70%	0.85
Articles 80%	0.88
Articles 90%	0.90
Articles 100%	0.91

TABLE 5.3: Evaluation results of the Context User Model with the creation of the user model considering 10% to 100% of classified articles.

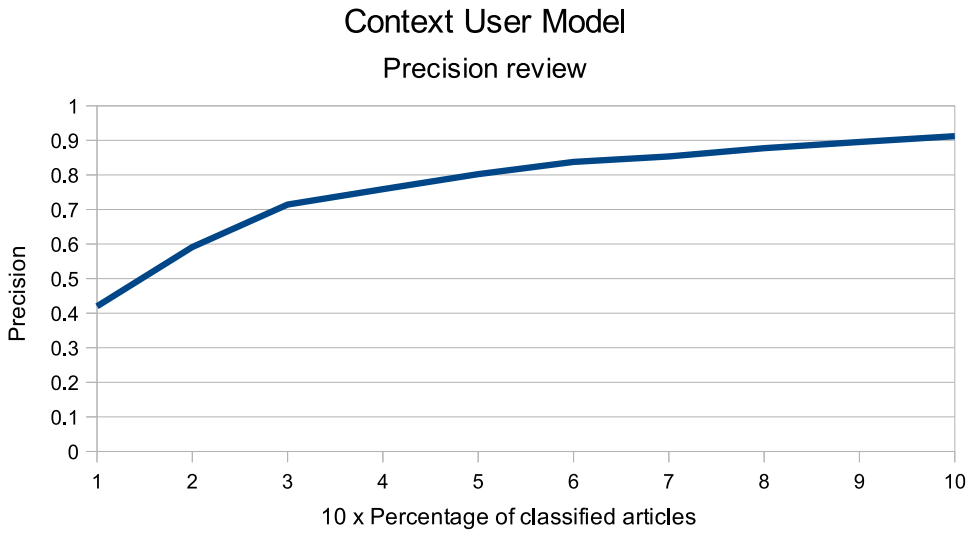

FIGURE 5.3: Data analysis of the recommendations according to the Context User Model.

messages and compares every one of them with new articles until it finds the best match. This constant searching and comparison is the mechanism that let the Context User Model learn. However, this learning process suggests that the Context User Model is not scalable[3]. Figure 5.3 shows a graphical result of the Context User Model evaluation and its logarithmic growing. The figure concludes that the learning process of the Context User Model is effective when a big number of classified messages are used to build the user model.

[3]However, in this thesis scalability is not a requirement.

5.2.2.3 Common User Model

The Common User Model evaluation considers the result of the average values of 7 different users. Table 5.4 shows the result of this evaluation.

Common User Model	Precision
Articles 10%	0.12
Articles 20%	0.32
Articles 30%	0.04
Articles 40%	0.46
Articles 50%	0.34
Articles 60%	0.31
Articles 70%	0.24
Articles 80%	0.69
Articles 90%	0.30
Articles 100%	0.52

TABLE 5.4: Evaluation results of the Common User Model with the creation of the user model considering 10% to 100% of classified articles.

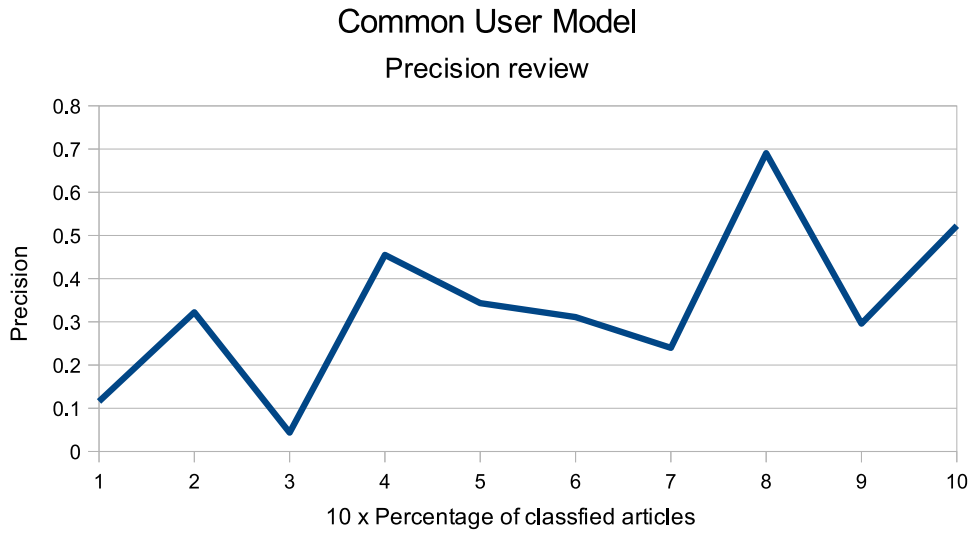

FIGURE 5.4: Data analysis of the recommendations according to the Common User Model.

According to table 5.4, recommendations regarding the Common User Model are very unpredictable. However, its tendency is to grow. At some points, its precision reaches almost 70 percent and, in the end, it reaches 52 percent. A possible reason of this unpredictable behaviour is that the Complementary User Model is created from the Basic User Model and, therefore, the Basic User Model of each user is not the same when more articles are used to create the model. This variability explains that users do not have the same preferences during the entire evaluation. Figure 5.4 shows a graphical way

of how this user model behaves as long as the users classify more messages. Although the Common User Model provides variable recommendations, its learning process suggests that as soon as the user classifies more articles, more precision the model will have. This observation can be made due to the user model has a precision of 52 percent when 100 percent of the classified messages are used to create the Common User Model.

5.2.2.4 Extended User Model

The Extended User Model evaluation considers the results of the average values of 7 different users. Table 5.5 shows the obtained results and, in addition, figure 5.5 shows the graphical results of this evaluation.

Extended User Model	**Precision**
Articles 10%	0.20
Articles 20%	0.32
Articles 30%	0.38
Articles 40%	0.41
Articles 50%	0.40
Articles 60%	0.38
Articles 70%	0.37
Articles 80%	0.33
Articles 90%	0.41
Articles 100%	0.41

TABLE 5.5: Evaluation results of the Extended User Model with the creation of the user model considering 10% to 100% of classified articles.

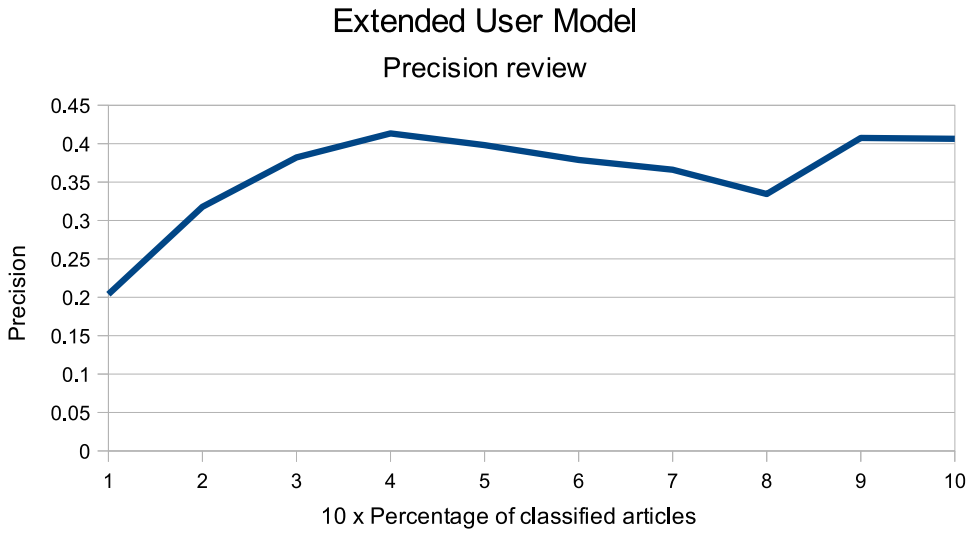

FIGURE 5.5: Data analysis of the recommendations according to the Extended User Model.

According to table 5.5, the precision of the recommendations, regarding the Extended User Model, improves until 40 percent of classified articles are used to create the model. After 40 percent, it sightly decreases but at 80 percent it starts to increase again. The reason for this result is that the Extended User Model uses the representation provided by all the presented user models, and therefore, it analyses more aspects of the user interests than any user model separately. This characteristic makes the Extended User Model adopt similar behaviours from other user models. Its learning process is very limited since the precision obtained at 40 percent of classified articles and the precision obtained at 90 and 100 percent is the same. One possible solution for the limited learning process would be to consider more the results of the Context User Model when the aggregation of the user models is analysed by the Extended User Model. This solution can be implemented since the evaluation of the Context User Model demonstrated that its learning process is more effective.

5.2.2.5 User Models' Comparisons

This section shows a comparison of the different user models results. This comparison allows to understand how precise the user models are regarding other user models. Table 5.6 shows an overview of the results presented in the sections above considering the 10 percent to 100 percent of the classified articles to create the user models.

	Basic	**Complementary**	**Context**	**Common**	**Extended**
Articles 10%	0.16	0.21	0.42	0.12	0.20
Articles 20%	0.26	0.26	0.59	0.32	0.32
Articles 30%	0.33	0.31	0.71	0.04	0.38
Articles 40%	0.35	0.31	0.76	0.46	**0.41**
Articles 50%	0.33	0.30	0.80	0.34	0.40
Articles 60%	0.31	0.29	0.84	0.31	0.38
Articles 70%	0.36	0.31	0.85	0.24	0.37
Articles 80%	0.31	0.27	0.88	**0.69**	0.33
Articles 90%	0.37	**0.36**	0.90	0.30	**0.41**
Articles 100%	**0.38**	0.35	**0.91**	0.52	**0.41**

TABLE 5.6: Comparison 10% to 100% of classified articles.

Table 5.6 presents the overall results of the objective evaluation. The table shows the precision of the presented user models regarding the percentage of classified articles used to create the user models. According to this information, the following observation can be noticed: all the user models, reach their best precision when the number of classified articles that are used to create the models are between 80 and 100 percent,

which indicates that as soon as the users classify more articles, the recommendation will be more precise.

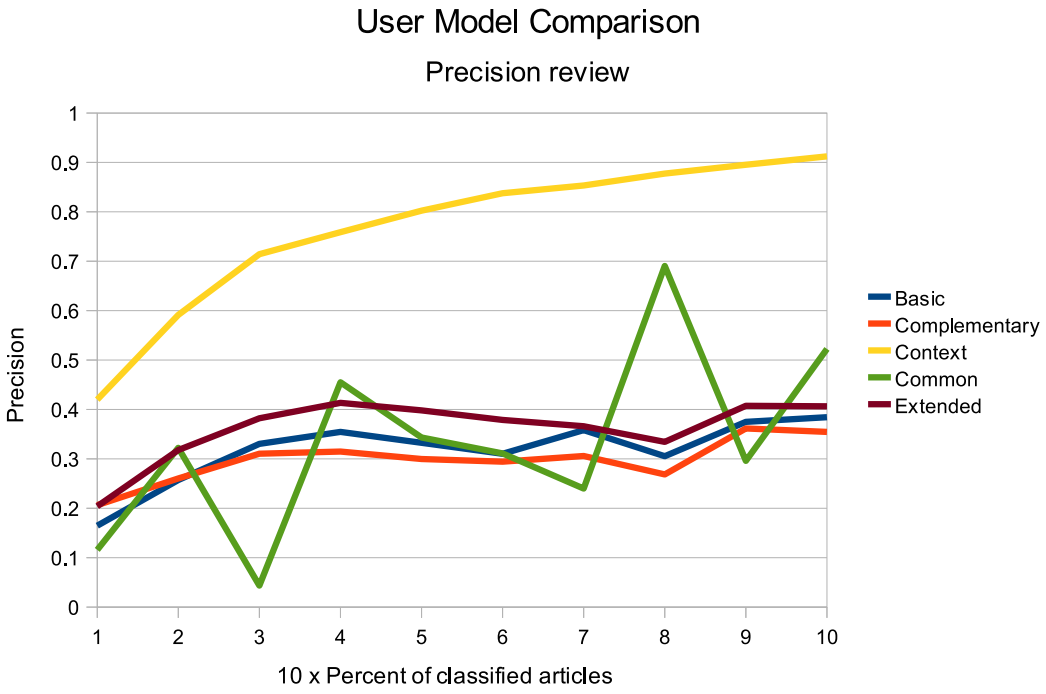

FIGURE 5.6: Data analysis of the recommendations according to the Complementary User Model.

Figure 5.6 presents an overview of the results in a graphical way. In this figure, it can be seen that the Context User Model has more precision during the entire evaluation. In contrast to this, the Complementary User Model is the model that has lower precision during the whole analysis. However, as it has been explained in the evaluation section 5.2.2.1 *Complementary User Model*, this model represents concepts that are not necessary in the articles that the user has classified.

It is also observed that there is a trade-off between stability and precision. For example, the Common User Model shows high precision at some points but also very low, like at 30 and 70 percent of classified articles. The Basic User Model and the Complementary User Model show more stability but their precisions are low, in comparison to other user models. The Extended User Model tries to cope better this trade-off by reaching precisions of 41 percent. The Context User Model shows more desired results but its limitation is that it cannot provide alternative recommendations like the Complementary User Model.

The next section will present another approach to evaluate user models that is more adequate to analyse the Complementary User Model and the Context User Model.

5.3 Real Users' Evaluation

This evaluation consists of the analysis of 100 different articles classified by 7 different users. The real users' evaluation considers the 100 percent of the messages classified by a user. According to the user's classification, the recommender system builds the user models and recommends new articles that have not been classified. This method represents a more realistic evaluation due to the users decides whether the recommendations have been done properly or not.

To evaluate the real users' interests, users evaluated the results given by the Basic User Model, Complementary User Model, Context User Model, Common User Model and Extended User Model separately. Each user model, except the Context User Model, has considered the threshold set in the previous evaluation and the user interface displays 30 of 110 new articles. Regarding the threshold utilized by the Context User Model, it has been set to 0.1 due to the maximum value reached in this evaluation was 0.2. This threshold was set according to what has been explained in section 5.2, *Objective Evaluation*. To have a point of comparison for this evaluation, 30 new articles have been randomly displayed to the user to evaluate the precision of this basic approach.

Table 5.9 presents the results of the evaluations obtained from 7 different users. In this analysis, are shown the number of interesting articles for a determined user in a set of 30 different articles recommended by the user models.

Username	**Random**	**Basic**	**Compl.**	**Context**	**Common**	**Extended**
User 01	9	20	21	27	22	28
User 02	14	23	25	27	30	29
User 03	8	10	18	22	22	26
User 04	7	14	18	21	20	20
User 05	28	24	21	24	25	21
User 06	17	20	25	23	19	22
User 07	13	18	21	24	23	25
Aver. art.	14	18	21	24	23	24
Total art.	30	30	30	30	30	30
Precision	0.46	0.61	0.71	0.80	0.77	0.81

TABLE 5.7: Average and accuracy of according to users' criteria.

The results presented in table 5.7 conclude that the precision of the user models are greater than the precision obtained in the first evaluation. The reason for that is that both evaluation analyses different aspects of the user models. The first evaluation analyses the learning process of the user models and the second evaluation analyses the

precision according to the user criteria. Furthermore, the precision of this evaluation is calculated differently as the precision formula presented in the previous section. In this evaluation the precision has been redefined to $Precision = r/t$, where r is the number of articles where the user agrees that are correctly recommended and t is the total number of articles provided by the recommender system[4].

According to the results provided in table 5.7, the following conclusions can be determined: all user models but the Basic User Model provide a precision over 70 percent. One reason for the low precision of the Basic User Model is that the model evaluates whether a new article is recommendable or not only according to what the user has already classified. However, this model does not represent what the users may be interested beyond of what he or she has already classified.

In contrast to the Basic User Model, the Complementary User Model provides additional information due to the user model has also words related to what the user has classified. This feature of the Complementary User Model let recommender to provide alternative articles to the user. The Context User Model let the recommender to search for the most similar article according to what he or she has already classified. A possible reason of why the Context User Model has high precision is due to the greatest similarity found between two articles is around 10 percent. This "low" similarity is close enough to recommend similar articles with different topics but with the same semantic terms.

The Common User Model provides recommendations regarding the preferences of other users with similar preferences. This allows the recommender to provide a wider domain of articles to the user and not only a bounded number of them. Finally, the Extended User Model analyses all the user models to provide its own recommendations. This approach considers the good aspects of the user models but also the bad ones, however, the evaluations conclude that the Extended User Model has a better trade-off between stability and precision and presents higher precision, according to the users' decisions, than the other user models separately.

Once that the evaluations have been performed and analysed, the suitability of the presented user models can be discussed. Firstly, although this has not been evaluated, it is very likely that the user models can be used in any case where text analysis is required. This affirmation is based on the approach that is used to create the user models.

Regarding the usability of each user model, the Basic User Model is necessary since the Complementary User Model and the Common User Model are created from this

[4]In this evaluation, the number of recommendations was 30.

model. The Complementary User Model and the Common User Model can be used in systems where different items alternatives are needed, such as news portals and feed readers. The Context User Model is the only user model that does not depend on the Basic User Model. Its advantage is that it is capable to recommend similar items due to it uses a similarity coefficient. However this characteristic makes the Context User Model very domain specific, where to recommend alternative articles with different concepts are not needed, for example, emails or activity streams. Finally, the Extended User Model provides higher precision[5] and more stability.[6]

These characteristics make the Extended User Model a good candidate for systems that require recommendations with high precision and alternative recommendations. In addition, the Extended User Model can interpret the results of other user models in different ways. These interpretation possibilities make the Extended User Model more flexible and suitable for any kind of scenarios.

5.4 Summary

This section presented the results of the evaluations done by 7 users who classified 100 different articles from a technology web log[7]. Different user models were evaluated using the analysis of precision according to the percentage of classified articles to create the user models. The evaluations were performed in two different ways: objective evaluation and real users' evaluation.

The objective classification evaluation measures the precision and evaluates whether the recommender system, with the presented user models, is working properly or not. It also evaluates the learning process of user models and presents a comparison between them. Regarding the results, the user models reach their maximum precision when the user models are created with a 80 to 100 percent of the classified articles. According to the comparison presented by this evaluation, the Context User Model provides higher precision and more stability during the whole evaluation. However, the Context User Model does not provide alternative recommendations. If this feature is required, a trade-off between stability and precision happens. To solve this problem, the Extended User Model copes with this trade-off by increasing the precision, regarding the Basic User Model and the Complementary User Model, and the stability of the Context User Model.

[5] According to the real users' evaluation.
[6] According to the objective evaluation.
[7] http://www.engadget.com - Last visit July 2011.

The real users' evaluation showed different results, however, this evaluation was not performed automatically like the objective evaluation, it was evaluated by users who decide whether the recommendations were effective or not. In this evaluation, the Extended User Model provided recommendations with higher precision than other user models due to this model tackles different aspects of user interests, such as context of the information and related concepts. Finally, a discussion about the usability of the presented user models was introduced.

Chapter 6

Epilogue

The realization of this thesis was motivated by the task of modelling users' interests. In this work, the following chapters were presented:

Chapter 1, *Introduction*, described the importance of the addressed task and outlined research questions that look for the improvement of user modelling. This chapter presented also the use cases and the requirements to fulfil the goals of this thesis.

Chapter 2, *Background and State of the Art*, covered background information about user modelling and recommender systems. It also defined the necessary methods and procedures to calculate recommendations regarding different user models. The chapter also introduced several state of the art systems that fulfil some of the presented requirements and a comparison between them was done to have an overview of their limitations.

Chapter 3, *Description of the Concept*, introduced the concept of the Extended User Model, a set of user models that fulfil the requirements introduced in the first chapter. These requirements were fulfilled by the Complementary User Model, that provides related concepts to the Basic User Model. The Context User Model, that analyses the context of messages that the user has classified. The Complementary User Model, that represent the user interests of users with same similarities. This chapter explained the architecture of the recommender system and how these models are created to be used by the system.

Chapter 4, *Implementation*, covered the details about the implementation of the recommender system. The package structure and the classes were explained. In addition, other aspects, such as the development languages and the user interface, were discussed.

Chapter 5, *Evaluation*, evaluated the implementation of the recommender system. For the evaluation, a set of articles was classified by 15 different users. The chapter evaluated the implementation of the recommender system in two ways: predefined classification evaluations and real users' interests evaluation. The first evaluation analyses the precision of the user models and the learning process of them. The second evaluation shows the results of the recommendations regarding new articles, where the user had to decide whether the recommended article is liked or disliked.

6.1 Conclusion

This section reviews the goals and the specific requirements presented in chapter 1, *Introduction*, according to the results obtained in the evaluation chapter of this work. The section discuss each goal separately to analyse different aspects and solutions in order to consider what can be improved in the future.

Although the obtained results of the evaluations, the presented concept introduces a new way to represent different aspects of user interests. In addition, recommendations that are most accurate according to the standard metrics are sometimes not the recommendations that are most useful to users. Accuracy metrics cannot check different situations because they are designed to judge the accuracy of individual item predictions; they do not judge the contents of entire recommendation lists. Unfortunately, it is these lists that the users interact with. All recommendations are made in the context of the current recommendation list and the previous lists the user has already seen. The recommendation list should be judged for its usefulness as a complete entity, not just as a collection of individual items[16].

Regarding the **supportability for a number of user models**, the framework, in this implementation, supports 4 user models, i.e. the Basic User Model, Complementary User Model, Context User Model and the Common User Model. In order to evaluate further user models, the user model definition must be programmed in order to be read and executed by the Profile Creator. However, the framework architecture is simple and let easily add more user models and set different recommendations criteria.

In order to **recommend unclassified items**, the recommender system analyses the text that is contained in the item. This content extraction approach let the system to create user models by representing the text that is in the item itself without considering some meta-data or priori classification that an item may have.

To **recommend related items**, different approaches have been utilized. In one hand, a hierarchy of terms has been implemented using the Domain Words Model. This model

let the Complementary User Model to find related words to a specific terms. If related words are in different items, it could mean that those items may be interest to the user. Regarding the real users' evaluation, the Complementary User Model has a precision of 71 percent approximately. To tackle this goal, also the Common User Model has been implemented in order to recommend related items. This model offers to the user the possibility to extend the recommendations alternatives. The same evaluation shows that the Common User Model provides a precision of 77 percent.

According to the **domain text extensibility**, the goal has been achieved by, again, analysing the content of the item. In the common case, different domains have different ways to classify and rate items, however, because of the mentioned approach, the user models store terms that are contained in the text of the items. Therefore, any item that contain text can be modelled by the presented user models.

Regarding the **flexibility**, a complete evaluation was performed to determine the flexibility of the user models. The evaluation results determine that the user models increase their precision after some amount of time. However, their precision start to decrease sightly in some cases and fast in others. This is due to the created user model turns too specific, which means for the items less probability to be selected due to its low probability. To solve this problem, all the results provided by the user models were considering by the Extended User Model, which has less precision decrease when more items are classified.

The **Adaptability** of the user models is addressed by the timestamp. The timestamp let the user model to adapt when the users' preferences change by removing the terms with less importance after some period of time. It has also been described two kinds of adaptability: adaptation performed in short and long term.

In chapter 5, *Evaluation*, the **user models' differences** were introduced through the comparison between them in both evaluations: objective evaluation and real users' evaluation. The differences have been presented regarding their precision and the number of classified articles used to create the user models. Graphical chats have been also provided to understand the user models' behaviour and to determine the precision's difference at different points.

Finally, according to the **user models suitability**, the chapter 5, *Evaluations*, also described different cases where the presented user models can be used.

6.2 Future Work

During the research of this work, a number of ideas have been raised that can be considered in a future work. One of these ideas is to investigate different alternatives in order to create a more scalable recommender system. In addition, further research is needed in order to increase the performance of the presented user models.

Furthermore, additional investigation is needed to automatize an hierarchy structure in order to provide related words according to a concept. In this work, the Domain Word Model was used to add additional information to the Complementary User Model. The Domain Word Model was created by adding words manually to a Java class, however, this action could be performed automatically by the system. As long as new terms are showing up, such as new products or services from a company, these words should be recognized and added to the correct concept. This would allow the Domain Word Model learn automatically and provide a more complete list of related words to the Complementary User Model.

Considering other aspects of this concept, other contexts of a message could be identified. For example, someone who works in a company A would like to know about the current crisis of a competitor company B. In this case, this user would be interested in such articles in order to take advantage of the situation of his or her competitor. In current recommender systems, these contexts have not been identified. Therefore, further investigation will be performed to identify these aspects in order to provide recommendations regarding these contexts.

Bibliography

[1] E. Santos Jr., H. Nguzen, Q Zhao and H. Wang *User modeling for intent prediction in information analysis*, Department of Computer Science and Engineering, University of Connecticut.

[2] A. Kodakateri, S. Gauch, H. Luong, J. Eno *Conceptual Recommender System for CiteSeer*, RecSys 2009, October 23-25. New York, USA. ACM, 2009.

[3] E. Banos, I. Katakis, N. Bassiliades, G. Tsoumakas and I. Vlahvas *PersoNews: A Personalized News Reader Enhanced by Machine Learning and Semantic Filtering*, Department of Informatics, Aristotle University of Thessaloniki, Greece. Springer-Verlag Berlin Heidelberg 2006.

[4] L.G. Perez, L. Martinez, F. Mata, M. Barranco *Un modelo de recomendacion con perfiles de usuarios linguisticos*, Departamento de Informatica, Universidad de Jaen.

[5] A. Das, M. Datar, A. Garg *Google News Personalization: Scalable Online Collaborative Filtering*, WWW 2007, May 8-12, Alberta Canada. ACM 2007.

[6] J. Oh, S. Lee, E. Lee *A User Modeling using Implicit Feedback for Effective Recommender System*, International Conference on Convergence and Hybrid Information Technology, IEEE 2008.

[7] S. M. Galan Nieto *Filtrado Colaborativo y Sistemas de Recomendacion*, Universidad Calos III de Madrid, IRC 2007.

[8] M. Garden and G. Dudek *Mixed Collaborative and Content-Based Filtering with User-Contributed Semantic Features*, McGill University, Centre for Intelligent Machines. University St. Montreal, Quebec, AAAI 2006.

[9] N. K. Nagwani1 and A. Bhansali *An Object Oriented Email Clustering Model Using Weighted Similarities between Emails Attributes*, International Journal of Research and Reviews in Computer Science, 2010.

[10] E. Vozalis, K. G. Margaritis *Analysis of Recommender Systems' Algorithms*, University of Macedonia, Thessaloniki, Greece.

[11] H. Pang, L. Zhou, H. Liu *Personalization Recommendation Service in Enterprise Information Portal*, IEEE, 2010.

[12] C. Drouin, D. Brown *Design and Development of a Learning-Augmented RSS Feed Reader Program The Mindful Reader*, Worcester Polytechnic Institute, Massachusetts, USA.

[13] Tom M. Mitchell *Machine Learning*, McGraw Hill.

[14] J. Dean, S. Ghemawat *MapReduce: Simplified Data Processing on Large Clusters*, San Francisco, CA. 2004.

[15] O. David, D. Dursun *Advanced Data Mining Techniques*, Springer, 1 edition. 2008.

[16] S. M. McNee, J. Riedl, J. A. Konstan *Accurate is not always good: How Accuracy Metrics have hurt Recommender Systems*, ACM, 2006.

[17] F. Mendonca, A. Ozaki *Comparison of text sets using Data Mining and Similarity Measure Methods.*

[18] H. Bruecher, G. Knolmayer, M. Mittermayer *Document Classification Methods for Organizing Explicit Knowledge*, University of Bern, 2002.

[19] A. Kobsa *Generic User Modeling Systems*, University of California, Irvine, 2002.

[20] B. Kules *User Modeling for Adaptative and Adaptable Software Systems*, University of Maryland, 2000.

[21] T. Hofmann *Probabilistic Latent Semantic Analysis*, University of California, 2004.