

ONTOLOGY FILTERING

THÈSE N° 3934 (2007)

PRÉSENTÉE LE 19 OCTOBRE 2007

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE D'INTELLIGENCE ARTIFICIELLE
PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Vincent SCHICKEL - ZUBER

ingénieur informaticien diplômé EPF
et de nationalité française

acceptée sur proposition du jury:

Prof. E. Telatar, président du jury
Prof. B. Faltings, directeur de thèse
Prof. A. Boyer, rapporteur
Prof. B. Smyth, rapporteur
Prof. A. Wegmann, rapporteur



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2007

Abstract

With the ever growing importance of internet, people are becoming overwhelmed by information. More concretely, consider a situation where you find yourself with an evening alone and would like to rent a DVD to watch. For several reasons, this is a difficult problem. First, most people have limited knowledge about the alternatives. Second, the set of alternatives changes frequently. In our example, there could be thousands of movies to choose from, and new movies are released on a daily basis. Third, this is an example of a low user involvement decision process, where the user is not prepared to spend hours expressing her preferences. *Recommender systems* have been designed to help people in this situation by finding the most relevant items based on the person's preferences. Two kinds of techniques are used in eCommerce sites today.

The most widely used technique is collaborative filtering, (CF), which recommends products to users based on the experience of others. Amazon.com, with its 29 million customers and several million catalog items, uses item-based collaborative filtering to recommend items to the user. Item-based collaborative filtering works by finding similar items to the ones rated by the user, and then combines those similar items into a recommendation list. Thus, the ability of CF to recommend items depends on the capability of identifying a set of similar users. Furthermore, it does not build an explicit model of the user's preferences. Instead, preferences remain implicit in the ratings that the user gives to some subset of products, either explicitly or by buying them. Despite its popularity, CF suffers from profound problems such as the cold-start problem, and scalability issues. The former problem is due to the fact that a user must rate (too) many items before getting a recommendation, while the latter is correlated to the complexity of finding and working with the list of similar users.

The other technique is the preference-based approach, (PBA). Here, a user is asked to express explicit preferences for certain attributes of the product. If preferences are accurately stated, then multi-attribute utility theory provides methods for finding the most preferred product, even when the set of alternatives is extremely large and/or volatile. Preference based approaches do not suffer from the same problem as collaborative filtering. However, the user needs to express a potentially quite complex preference model. This may require a large number of interactions, and places a higher cognitive load on the user since he has to reason about the attributes that model the product. To be able to reason over the items, multi-attribute utility theory also requires that products are modeled by a set of well defined attributes, which is sometime impossible to obtain in real life situation.

Despite being very popular, these recommender systems sometimes fail to achieve high recommendation accuracy in eCommerce environments; especially when users' preferences are rare. Our analysis shows that these problems are due to two fundamental issues. First, current recommender systems use inappropriate models of the items and of the users' preferences. Second, recommender systems must elicit too many preferences from the user in order to build the user's preference profile.

This dissertation proposes the ontology filtering approach that can overcome most of the problems faced by previous approaches, while achieving a better prediction accuracy than item-based collaborative filtering. The intuition behind ontology filtering is that the information captured by the topology of an ontology can be used to estimate missing preferences. The main novelties of this technique are to model the content of the eCatalog and infer missing preferences using the ontology, and then use the inferred information for directly recommending items to the user.

Keywords: Recommender Systems, Ontology, Preferences, Ontology Filtering.

Résumé

Avec l'essor constant d'Internet, les gens deviennent littéralement submergés par l'information. Pour illustrer ce problème plus concrètement, imaginez une situation où vous vous trouvez seul pour la soirée, et que vous voudriez louer un DVD pour tuer le temps. Ce problème est difficile à résoudre, et il y a plusieurs raisons pour cela. Tout d'abord, les gens n'ont que des connaissances très limitées sur les alternatives possibles. Deuxièmement, les alternatives changent fréquemment. Troisièmement, cet exemple est un problème de décision à faible risque où l'utilisateur n'est pas prêt à passer plusieurs heures pour exprimer ses préférences. Pour aider les gens dans cette situation, les *systèmes de recommandations* ont été créés afin de trouver les objets les plus pertinents d'après les préférences d'une personne. A ce jour, deux techniques sont couramment utilisées par les sites d'eCommerce.

La technique la plus utilisée est collaborative filtering, (CF), qui recommande des objets à un utilisateur d'après l'expérience d'autres utilisateurs. Avec ses 29 millions de clients et plusieurs millions de produits dans son catalogue, Amazon.com utilise item-based collaborative filtering pour recommander des objets à ses utilisateurs. Item-based collaborative filtering fonctionne de la manière suivante. Tout d'abord, une liste d'objets similaires à ceux achetés auparavant par l'utilisateur est créée. Ces objets sont ensuite combinés dans une liste de recommandations. Par conséquent, les recommandations proposées par CF dépendent de sa capacité à identifier l'ensemble des objets similaires. De plus, CF ne construit pas explicitement un modèle des préférences de ces utilisateurs. En réalité, les préférences restent implicitement cachées dans les votes que les utilisateurs ont exprimés sur un sous-ensemble des produits; soit explicitement ou en les achetant. Malgré sa popularité, CF souffre de plusieurs gros problèmes comme le démarrage à froid et l'extensibilité du système. Le premier problème est dû au fait qu'un utilisateur doit voter sur beaucoup (trop) d'objets avant d'obtenir des recommandations, tandis que le deuxième est la conséquence de l'utilisation des proches voisins.

L'autre technique s'appelle preference based approach (PBA). Dans ce cas, le système demande explicitement à un utilisateur de donner ses préférences sur certains attributs modélisant un objet. Si ces préférences sont correctement données, alors la théorie sur l'utilité des attributs multiples donne des outils permettant de trouver facilement les meilleurs objets, même quand le nombre d'alternatives est extrêmement large et/ou volatile. PBA ne souffre pas des mêmes problèmes que collaborative filtering. Par contre, un utilisateur doit fournir un modèle de préférence souvent complexe. Ceci peut conduire à de multiples interactions entre le système et l'utilisateur, et un accroissement significatif du poids cognitif vu

qu'un utilisateur doit raisonner sur les attributs modélisant les objets. De plus, la théorie sur l'utilité des attributs multiples stipule que chaque objet doit être modélisé par un ensemble bien défini d'attributs, ce qui n'est bien sur pas toujours possible en pratique.

Malgré leurs popularités, les systèmes de recommandations actuels n'arrivent pas toujours à recommander des produits avec précision dans le contexte de l'eCommerce; tout spécialement quand peu de préférences sont disponibles. Notre analyse montre que ces difficultés sont dues principalement à deux problèmes. Premièrement, les techniques actuelles utilisent un modèle des objets ainsi que des préférences inappropriés. Deuxièmement, les systèmes de recommandations doivent obtenir trop de préférences des utilisateurs afin de construire leurs modèles.

Cette dissertation propose ontology filtering; une nouvelle technique de recommandation qui permet de résoudre la plus part des problèmes auxquels font face les solutions actuelles, tout en augmentant la précision des recommandations. L'intuition derrière ontology filtering est d'extraire l'information contenue dans une ontologie afin de l'utiliser pour estimer les préférences inconnues des utilisateurs. Les nouveautés de cette technique ne résident pas seulement dans le fait d'utiliser une ontologie pour modéliser les objets d'un eCatalog, deviner les préférences manquantes, et pour recommander directement les objets aux utilisateurs.

Mots-clefs: Système de recommandations, Ontologie, Préférences, Ontology Filtering, Le filtrage par ontologie.

Acknowledgements

I would like to address my sincerest thanks to Professor Boi Faltings, who gave me the opportunity to work at the Artificial Intelligence Laboratory. His helpful advices and comments allowed me to focus my research directions, and guide me through my phd journey. Moreover, the results which are presented in this dissertation would never have been possible without his support.

I am very grateful to Professors Anne Boyer, Barry Smyth, and Alain Wegmann for their careful reading of my thesis and the many valuable comments I received during and after the private thesis defense.

Thanks also to the members of the AI Lab and the Decision Group who have listened to my numerous presentations on ontology filtering and its components. Their comments have proven very valuable when writing conference papers and this thesis. I would also like to thank Professor Martin Columbus for his helpful insight in graph theory.

Remerciements vont bien sur à ma famille qui à su m'écouter, mais surtout laisser suivre mon instinct pendant ces 3 années de recherche.

Joëlle, je voudrais te remercier de tout mon coeur pour m'avoir soutenu pendant ces derniers mois. Ta confiance en moi m'a porté tout au long de ce travail, et ton aide précieuse m'a fait avancer à pas de géant. Merci!

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Motivation | 1 |
| 1.2 | Foundations in recommender systems | 5 |
| 1.3 | Problem definition and the proposed solution | 9 |
| 1.3.1 | Potential applications for ontology filtering | 10 |
| 1.3.2 | Applications not suitable for ontology filtering | 12 |
| 1.4 | Contributions | 13 |
| 1.5 | Overview of the dissertation | 13 |
| 1.6 | Intellectual property protection | 15 |
| 2 | Background | 17 |
| 2.1 | Evaluating recommender systems | 17 |
| 2.1.1 | Measuring the accuracy | 18 |
| 2.1.2 | Measuring non-obvious recommendations | 19 |
| 2.2 | Overview of recommender systems techniques | 19 |
| 2.2.1 | Non-personalized recommender systems | 23 |
| 2.2.2 | Personalized recommender systems | 24 |
| 2.3 | Content-based approaches | 29 |
| 2.3.1 | Information retrieval approach | 30 |
| 2.3.2 | Machine learning approach | 33 |
| 2.3.3 | Preference-based approach | 36 |
| 2.4 | Collaborative filtering | 40 |
| 2.4.1 | Memory-based collaborative filtering | 41 |
| 2.4.2 | Model-based collaborative filtering | 43 |
| 2.5 | Knowledge-based recommender systems | 49 |
| 2.5.1 | Conversational recommender systems | 49 |
| 2.6 | Usage of taxonomies in recommender systems | 51 |
| 2.7 | Collaborative filtering vs. Preference-based approach | 53 |
| 2.8 | Ontology filtering | 55 |
| 3 | Modeling eCatalogs with ontologies | 57 |
| 3.1 | Introduction | 57 |
| 3.1.1 | Ontology vs taxonomy | 59 |
| 3.2 | Usage of ontologies in the Web | 60 |
| 3.3 | The multi-hierarchical structure of an ontology | 61 |

| | | |
|----------|--|------------|
| 3.3.1 | Properties of the score of a concept | 62 |
| 3.3.2 | The a-priori score of a concept | 64 |
| 3.3.3 | Analogy with existing work | 67 |
| 3.3.4 | The ontology | 69 |
| 3.4 | The eCatalog ontological model | 70 |
| 4 | Modeling the users' preference profiles | 73 |
| 4.1 | Introduction | 73 |
| 4.2 | Eliciting the user's preferences | 74 |
| 4.3 | The user's preference profile | 75 |
| 5 | Inferring missing user's preferences | 79 |
| 5.1 | Introduction | 79 |
| 5.2 | From user's profile to user's ontological profile | 80 |
| 5.3 | Inference from one concept to another | 83 |
| 5.3.1 | Finding the lowest common ancestor | 84 |
| 5.3.2 | Upward inference | 86 |
| 5.3.3 | Downward inference | 87 |
| 5.3.4 | Upward & downward inference | 88 |
| 5.4 | Finding the closest concept | 89 |
| 5.4.1 | Existing measures of similarity | 90 |
| 5.4.2 | Transferring the score from one concept to another | 91 |
| 5.4.3 | Getting a lower bound value | 92 |
| 5.4.4 | The ontology structure based similarity - OSS | 93 |
| 5.4.5 | Example | 93 |
| 5.5 | The overall inference process | 94 |
| 5.5.1 | Example | 95 |
| 5.6 | Recommending items to a user | 96 |
| 5.6.1 | Example | 97 |
| 6 | Learning the structures of the ontologies | 99 |
| 6.1 | Introduction | 99 |
| 6.2 | Clustering algorithms | 100 |
| 6.3 | Learning a set of hierarchical taxonomies | 103 |
| 6.3.1 | Selecting the best learnt eCatalog ontological model | 105 |
| 6.4 | Learning multi-hierarchical taxonomies | 107 |
| 7 | Architecture of ontology filtering | 109 |
| 7.1 | Introduction | 109 |
| 7.2 | The architecture | 110 |
| 7.2.1 | The presentation layer | 112 |
| 7.2.2 | The application layer | 112 |
| 7.2.3 | The data layer | 114 |

| | | |
|----------|---|------------|
| 8 | Experimental results | 117 |
| 8.1 | Validation of the model | 117 |
| 8.1.1 | Experiment I - WordNet | 118 |
| 8.1.2 | Experiment II - GeneOntology | 120 |
| 8.2 | Ontology filtering with a static ontology | 123 |
| 8.2.1 | Experimental set-up | 123 |
| 8.2.2 | Experiment III - Behavior of ontology filtering | 126 |
| 8.2.3 | Experiment IV - Ontology filtering vs other techniques | 127 |
| 8.3 | Ontology filtering with the learnt taxonomies | 130 |
| 8.3.1 | Experimental set-up | 130 |
| 8.3.2 | Experiment V - Evaluating the item-to-item similarity metrics | 132 |
| 8.3.3 | Experiment VI - Evaluating Algorithms 7 and 8 | 133 |
| 8.3.4 | Experiment VII - Behavior of multi-hierarchical ontologies | 139 |
| 8.3.5 | Experiment VIII - Ontology filtering vs collaborative filtering | 141 |
| 8.4 | Discussion | 144 |
| 9 | Conclusions | 147 |
| 9.1 | Scope | 147 |
| 9.2 | Contributions | 148 |
| 9.3 | Limitations | 150 |
| 9.4 | Future research | 151 |
| 9.5 | Conclusion | 151 |
| A | Classification of recommender techniques | 153 |
| B | Transitivity of OSS | 155 |
| C | Ontology filtering: a scenario | 157 |
| C.1 | The scenario | 157 |
| D | Existing ontologies | 167 |
| D.1 | WordNet | 167 |
| D.2 | The GeneOntology | 168 |
| D.2.1 | Cellular component | 170 |
| D.2.2 | Biological process | 170 |
| D.2.3 | Molecular function | 170 |
| D.2.4 | Ontology structure | 170 |
| E | Detailed experimental results | 173 |
| E.1 | WordNet | 173 |
| E.2 | Predefined ontology vs learnt ontologies | 175 |
| E.3 | Improvement of OF over CF | 176 |
| F | Curriculum Vitae | 189 |

List of Figures

| | | |
|------|---|----|
| 1.1 | The three primary usage of a World Wide Web browser in 1995 | 4 |
| 1.2 | The three tier architecture | 5 |
| 1.3 | Basic component of a recommender system following the 3-tier paradigm . | 6 |
| 1.4 | The recommendation process of a recommender system | 8 |
| 1.5 | US business-to-consumer revenues in US\$ billions over the past five years. Data was obtained from the annual surveys by Forrester Research for Shop.org | 11 |
| 1.6 | Layout and dependencies between each chapters in this dissertation | 16 |
| 2.1 | Illustration of the precision and recall metrics | 18 |
| 2.2 | Main techniques used in recommender systems | 22 |
| 2.3 | This snapshot illustrates the non-personalized recommender system of Moviefinder.com, where a news item is recommended by average ratings of previous readers. | 23 |
| 2.4 | This snapshot illustrates the non-personalized recommender system of Digg.com, where a news item is recommended by the maximum number of diggs as- signed by previous readers. | 24 |
| 2.5 | A snapshot of value elicitation from Notebookreview.com | 25 |
| 2.6 | A snapshot of rating based preference elicitation from Amazon.com. | 26 |
| 2.7 | An example of the critiquing approach from the recommender system Entree. | 27 |
| 2.8 | Three different views of the Apollo 13 movie: <i>structured</i> , <i>semi-structured</i> , and <i>unstructured</i> model. | 30 |
| 2.9 | The information retrieval process. | 30 |
| 2.10 | Illustration of the cosine angle between an item i and user u in a 2 dimen- sional space. | 32 |
| 2.11 | The naive Bayesian classifier process. | 34 |
| 2.12 | The preference-based approach process. | 38 |
| 2.13 | Illustration of the value elicitation process for the preference-based approach | 38 |
| 2.14 | Two utility functions associated to the price attribute of Figure 2.13; (a) shows the predefined utility function, and (b) shows the utility function after that a user said that she wants a computer less or equal than 1500\$. . . | 39 |
| 2.15 | The user-based collaborative filtering process. | 42 |
| 2.16 | Graphical aspect model of the movie domain, where P , Z , and M are random variables that respectively model the objects p , z , and m | 44 |
| 2.17 | Example of the item-based collaborative filtering for a user who is looking for the movie Apollo 13 | 46 |
| 2.18 | The item-based collaborative filtering process. | 47 |

| | | |
|------|--|-----|
| 2.19 | Illustration of the Entree recommender system. | 50 |
| 2.20 | Fragment from the Amazon.com taxonomy. | 51 |
| 2.21 | Approach used by Middleton et al. for reducing sparsity in collaborative filtering. | 53 |
| 2.22 | (a) matrix R , while (b) is matrix S computed from (a). | 55 |
| 3.1 | The models used to represent the items of an eCatalog by collaborative filtering and the preference based approach. | 57 |
| 3.2 | Graph that looks at two dimensions in a model: the ease of use for the user and the structure used for representing the items. | 58 |
| 3.3 | Two ontologies with a tree (a) and DAG (b) structure. | 61 |
| 3.4 | Two ontologies with implicit (a) and explicit (b) features. | 62 |
| 3.5 | A simple ontology to illustrate the a-priori score of the concept c | 65 |
| 3.6 | The behavior of the a-priori score of a concept c when the number of descendants varies from 0 to 50. The x-axis is the number of descendants of the concept c , while the y-axis measures the APS | 66 |
| 3.7 | A snapshot of the WordNet Ontology for red wine. | 67 |
| 3.8 | (a) a simple ontology λ and its APSs computed using Equation (3.4)(b). | 67 |
| 3.9 | Comparison of the APS and IC for concepts that have between 0 and 50 descendants. The concepts with 50 descendants is considered as the root of the ontology. | 68 |
| 3.10 | The ontology λ in DAML-OIL language. | 69 |
| 3.11 | Illustration of the eCatalog ontological model generated with Algorithm 1. | 71 |
| 4.1 | Distribution of the ratings on Amazon.com for the randomly picked CD "Mr A-Z". This figure has been extracted from [Hu et al., 2006]. | 75 |
| 4.2 | The two processes that can modify the user's preference profile. | 77 |
| 5.1 | Example of a user ontological profile for user toto. | 82 |
| 5.2 | Possible chains in a tree structure between the concepts x and y : (a) x is a child of y , (b) x is a parent of y , and x is neither a parent nor a child of y (c). | 83 |
| 5.3 | Example of paths for concepts h and i to the root concept a , where the path are in bold lines. | 84 |
| 5.4 | Illustration of the correspondences between $\alpha(x, y)$ and $\hat{\alpha}(x, y)$ | 87 |
| 5.5 | Illustration of the correspondences between $\beta(y, x)$ and $\hat{\beta}(y, x)$ | 88 |
| 5.6 | Illustration of the OSS approach. | 90 |
| 5.7 | (a) a simple ontology λ and its APSs (b). | 94 |
| 5.8 | (a) a simple ontology λ and its corresponding a-priori score (b). | 98 |
| 6.1 | Illustration of the first three clustering steps for the (a) partitional and (b) agglomerative clustering algorithm. | 101 |
| 6.2 | Illustration of the multi-hierarchical structure generated by Algorithm 9. | 107 |
| 7.1 | The three tier architecture with a thin-client. | 110 |
| 7.2 | The architecture of the ontology filtering recommender system. | 111 |
| 7.3 | Snapshot of the preference elicitation interface used in ontology filtering. | 112 |

| | | |
|------|---|-----|
| 7.4 | A possible entity-relationship schema for the eCatalog ontological model. | 115 |
| 7.5 | Example of a hierarchical relation in the eCOM, where concept x is the child of concept y | 115 |
| 8.1 | Illustration of the upwards and downwards distance in OSS. | 119 |
| 8.2 | An ontology modeling the sequence describing genes, where the hierarchy is arranged based on what the genes have in common. | 121 |
| 8.3 | Illustration of common subsequence of a concept c , $CS(c)$ | 122 |
| 8.4 | Hand made ontology for the MovieLens data set. | 124 |
| 8.5 | Illustration of the set up for experiments III and IV. | 125 |
| 8.6 | Ontology filtering versus other strategies for inferring the value functions and weights. | 127 |
| 8.7 | Ontology filtering versus other recommender strategies. | 128 |
| 8.8 | Novelty of various recommender techniques compared to the popularity approach. | 129 |
| 8.9 | Illustration of the set up for experiments V to VIII. | 131 |
| 8.10 | Number of extra clusters generated by steps 15 to 20 of Algorithm 9. | 139 |
| 8.11 | Accuracy of ontology filtering for the Jester data set when the ontology is generated from either the classical agglomerative clustering with <i>clink</i> , or with Algorithm 9 with φ set to 0.4. | 140 |
| 8.12 | Accuracy of collaborative filtering and ontology filtering for the Jester data set. For each recommender system, the experiment was run once with only 5 ratings to learn the model (5LS), and then a second time with 50 rated items (50LS). | 142 |
| 8.13 | Accuracy of collaborative filtering and ontology filtering for the Movilens data set. For each recommender system, the experiment was run once with only 5 ratings to learn the model (5LS), and then a second time with 50 rated items (50LS). | 143 |
| 8.14 | Average improvement in % of the F1 metric of OF over CF. | 143 |
| B.1 | (a) a simple ontology λ and its APSs (b). | 155 |
| C.1 | The identification page where user <i>Vincent</i> identifies himself. | 159 |
| C.2 | Ontology Filtering asks <i>Vincent</i> to either rate joke 17 or enter a joke of his choice. <i>Vincent</i> loves joke 17 and gives it 5 stars. | 160 |
| C.3 | The joke that is recommended to <i>Vincent</i> once the elicitation process is over. Joke 89 is recommended to <i>Vincent</i> as it is very close to joke 11 that he previously rated with 5 stars. | 161 |
| C.4 | Ontology filtering shows <i>Vincent</i> 's preferences. | 162 |
| C.5 | <i>Vincent</i> updates his preferences on joke 11 by changing the rating from 5 to 3 stars. | 163 |
| C.6 | The new recommendation that is made once <i>Vincent</i> has updated the rating of joke 11. | 164 |
| C.7 | <i>Vincent</i> updates all of his preferences with 5 stars to 4 stars. | 165 |

| | | |
|-----|--|-----|
| C.8 | Final recommendation made when no more preferences have a 5 star ratings. Note that the recommendation is now made using the third ontology, where the ontology is identified by the first digit of the identifier given to a concept. | 166 |
| D.1 | An extract of WordNet for the synset <i>red-wine</i> | 168 |
| E.1 | Ontology filtering with 5 items in the users' learning set, and using either the ontology generated by the thesis's author, or the set of 15 ontologies learnt using Algorithm 7. | 175 |
| E.2 | Ontology filtering with 50 items in the users' learning set and using either the ontology generated by the thesis's author, or the set of 15 ontologies learnt using Algorithm 7. | 176 |

List of Tables

| | | |
|-----|--|-----|
| 1.1 | Size of the World Wide Web in terabytes for the years 2000, 2003, and 2006. | 3 |
| 2.1 | Brief tradeoffs analysis between collaborative filtering and the preference based approach. | 54 |
| 4.1 | Example of the user preference profile for user u . | 77 |
| 5.1 | Distance between concepts x and z from the ontology found in Figure 5.7, where the distance is computed using Equation 5.21. | 94 |
| 5.2 | Toto's Ontological Profile constructed using Algorithm 5. | 96 |
| 5.3 | Items with their position in the list $ratings$ when Algorithm 6 is at step 8. | 98 |
| 6.1 | Example of attribute-value pairs for COBWEB. | 101 |
| 6.2 | Criteria functions used by distance-based clustering algorithms to merge and split clusters. | 103 |
| 8.1 | Correlation of various similarity metrics with human judgements collected by Miller and Charles on WordNet 2.0. | 118 |
| 8.2 | Different combinations of the coefficients α and β in OSS. The correlation is computed using the real human judgement from the Miller & Charles's experiment. | 119 |
| 8.3 | Correlation with human judgement when coefficient δ ranges from a value equals to the minimum APS in the ontology to 1. Note that under the assumption that $S(x) = 1/2$, then δ is equal to 2. | 120 |
| 8.4 | MAE of various similarity metrics on the three sub-ontologies of the GeneOntology: molecular Function (MF), biological process (BP), and cellular component(CC). | 121 |
| 8.5 | Average recommendation accuracy of CF and OF when using different similarity metrics to build the item-to-item similarity matrix S . The accuracy is measured by averaging the F1 value of all the users. | 133 |
| 8.6 | Notations used by the various algorithms in Experiment VI. | 134 |
| 8.7 | Relative F1 values obtained on the Jester dataset. The notation x, y means that users in U_{user} had respectively 5 and 50 ratings in their learning set LS . The number of leaf clusters ranges from 5 to 100. | 135 |

| | | |
|------|---|-----|
| 8.8 | Relative F1 values obtained on the MovieLens dataset. The notation x, y means that users in U_{user} had respectively 5 and 50 ratings in their learning set LS . The number of leaf clusters ranges from 5 to 1668. | 136 |
| 8.9 | Execution time T required for the clustering algorithm to generate the ontologies (in seconds). The element x of the tuple x, y is in seconds, while y measures the relative time rT from the best clustering algorithm (1.0 being the best). | 137 |
| 8.10 | Efficiency of the various clustering algorithms used to generate the ontologies, with the efficiency computed with Equation 8.5. | 138 |
| 8.11 | Brief tradeoffs analysis of ontology filtering; ¹ when learning the ontologies with Algorithm 7, ² when using multi-hierarchical ontologies, and ³ when using a small set of hierarchical ontologies. | 145 |
| A.1 | Sections where the various approaches in Table A.2 have been introduced. . | 153 |
| A.2 | Overview of the techniques used in Recommender Systems and their name given by various authors. | 154 |
| D.1 | Summary of the WordNet relations. | 167 |
| E.1 | Correlation of various similarity metrics with human judgements collected by Miller and Charles. | 174 |
| E.2 | P-values that measures the improvement of ontology filtering over collaborative filtering for the Jester data set. | 177 |
| E.3 | P-values that measures the improvement of ontology filtering over collaborative filtering for the MovieLens data set. | 177 |

List of Algorithms

| | | |
|---|--|-----|
| 1 | <i>Generating the eCatalog ontological model.</i> | 70 |
| 2 | <i>Building and retrieving a user's preference profile for a user u.</i> | 76 |
| 3 | <i>Building the user's ontological profile for a user u.</i> | 81 |
| 4 | <i>Finding the lowest common ancestor in a directed acyclic graph for a user u.</i> | 85 |
| 5 | <i>Building a complete user's ontological profile for a user u.</i> | 95 |
| 6 | <i>Recommending the top-N items to user u.</i> | 97 |
| 7 | <i>Learning a set of 15 distinct taxonomies, each having a hierarchical structure.</i> | 105 |
| 8 | <i>Selecting the best eCatalog ontological model from a set of possibilities eCOMs for user u.</i> | 106 |
| 9 | <i>Learning a Multi-Hierarchical ontology with a window size φ and threshold cluster θ for user u.</i> | 108 |

Main notations

| Symbol | Meaning of the symbol |
|-----------|---|
| OF | The ontology filtering recommender system. |
| CF | The collaborative filtering recommender system. |
| OSS | The ontology structure based similarity function. |
| F1 | The F1 accuracy metric. |
| MAE | The mean absolute error metric. |
| α | The coefficient of generalization for the upwards inference. |
| β | The coefficient of specialization for the downwards inference. |
| ρ | The coefficient of personalization for computing the hybrid score of a concept. |
| φ | The window size coefficient in the multi-hierarchical clustering algorithm. |
| θ | The number of leaf clusters in a taxonomy. |
| k | The number of neighbors used by collaborative filtering. |
| D | The distance function. |
| I | The set of items in the user-item matrix R , $I = \{i_1, i_n\}$. |
| R | The user-item matrix containing all the users' preference profiles. |
| S | The item-to-item similarity matrix. |
| U | The set of users in the user-item matrix R , where $U = \{u_1, u_m\}$. |

Chapter 1

Introduction

Users have two fundamental activities online: search and browse. When someone knows exactly what she is looking for, she searches for it. But when she is not looking for anything specific, she browses. The problem of searching an object on the web has been heavily studied in the past 15 years, and there is a 150 billion dollars winner - Google. However, good solutions for solving the browsing problem are yet to be found, but recommender systems are attracting a lot of attention from the research community.

Recommender systems have been designed to help people browse through a collection of item based on the preferences of the person and/or others. For several reasons, this is a difficult problem. For example, users usually have only limited knowledge of the content and structure of the items available, which limits what a recommender system can ask the user. Moreover, recommender systems are usually used in low user involvement decision process, where the user is not prepared to spend hours expressing her preferences. Beyond the words, the best example that shows the complexity of the problem is the 1 million dollar prize that the company Netflix¹ offers to anyone who can beat their recommendation system.

This dissertation believes that current recommender systems fail to meet users' expectations because of two fundamental problems: inadequate model of eCatalogs and elicitation overload. The former problem focuses on how a recommendation system models and reasons with the items it has to recommend; while the latter is the process of asking (too many) questions to the user in order to understand her preferences. This thesis introduces a new recommender system called ontology filtering that improves the recommendation accuracy by solving the two previously stated problems.

1.1 Motivation

We live in a world surrounded by information, far too much for a normal human being to process, even in a lifetime. In 1989, Richard Saul Wurman estimated that an average week-

¹www.netflix.com

day edition of *The New York Times* contained more information than the average person in seventeenth century England was likely to come across in a lifetime.

The Library of Congress in Washington is the world's largest library, and used to be the biggest source of information in the world. It was established on April 24, 1800, when President John Adams signed the bill establishing the federal government in Washington. It was initially funded with \$5000 that allowed the congress to acquire less than 1000 books. Today, the library contains over 134 million items, which require more than 500 miles of shelving. Among this massive collection, over 79 millions are books but only 20 millions are in fact catalogued in the Library of Congress classification system. Theoretically, it would take more than 216 thousands years for a person to read all the knowledge that has been acquired over the past 200 years.

However, in 1989, a revolution took place in Switzerland that changed the way we see information. Tim Berners Lee, a physicist at CERN, submitted a proposal² for sharing and editing information over the internet using hypertext. The *World Wide Web* was born. By November 1992, there were only 26 servers around the world, but the figure had increased to over 200 by October 1993. Until that day, the World Wide Web was primarily used by scientists, and was unknown to the general public. In 1993, it all changed when CERN announced that the World Wide Web would be free to anyone, and when NCSA released the first version of the web browser Mosaic. These two major events unlocked the web and made it available to anyone using PCs or Apple Macintoshes. Ten years later, the size of the World Wide Web was estimated³ to over 92 thousand terabytes, which roughly corresponds to over 230 times the amount of books in the Library of Congress.

From the beginning of the 60's until the web era, the challenge in computer science was how to store and efficiently access information. A lot of research and money has been invested into that domain, and led to what is commonly called today as *information systems*. The most common example of such systems are databases. A database is a collection of information organized in such a way that a computer program can quickly and efficiently access any piece of data. Databases allow users to store incredibly huge amounts of information very efficiently, which is then accessed using a Structured Query Language, (SQL). SQL is a computer language used to efficiently create, retrieve, update and delete data from a database system. Despite being very efficient and robust, SQL has the major inconvenient of being very rigid, and requires the user to have a complete knowledge of the underlying model in the database, along with advanced knowledge in the SQL.

With the rise of the World Wide Web, challenges in computer science have shifted from storing to finding the information. The web has also attracted users from all around the world, each having very different background, and a great majority of them had little knowledge about computers. Therefore, it was unfeasible to ask a user to query a database using a programming language such as SQL. This created the need for new tools. This is where search engines came to the rescue.

Search engines are web applications that take some keywords as input and return the documents that match these keywords. The main idea is to use the keywords contained in the documents as *indexes*. These indexes are then stored in databases, and allows docu-

²<http://info.cern.ch/Proposal.html>

³<http://www2.sims.berkeley.edu/research/projects/how-much-info-2003>

ments to be efficiently retrieved. Moreover, as the input are simple keywords rather than SQL query, this made search engines very easy to use by all user. As a consequence, search engines soon became very popular and many appeared on the web. Such examples are Altavista, Excite, Voila, Lycos and so forth.... but they soon became living - dead. The main reason lies in the quality of the returned documents. Due to the size of the web, most queries sent to search engines would return thousands, and even millions of results. Larry Page and Sergey Brin found a \$150 billion dollars solution - Google. As with other search engines, Google retrieves a set of document based on some keywords. The fundamental difference lies in the way the results are returned. Before returning some documents to the user, Google sorts the results based on a measure called *page rank*. For a given document d , page rank simply counts how many other documents reference d , and sorts the documents based on these relevance values. Thus, the first document shown to the user is the one that has the highest page rank value.

Despite Google's popularity and financial market strength, it is not the universal solution for helping users find information on the web. There are three reasons why google is not the ultimate solution:

1. Google only searches a small fraction of the web. The World Wide Web is actually composed of two parts: the *surface web* and the *deep web*. The surface web, also known as the indexable web, is the proportion of the World Wide Web that is indexed by search engines. Before a search engine can answer a query, it must copy the content of the web in its databases using web crawlers. Web crawlers are computer programs that for each web page it reads on the web, retrieves it, indexes it, and stores it in its database. To go to a next page, the web crawler simply follows hyperlinks found in documents. For various reasons such as password protection or dynamic hyperlinks, some documents cannot be reached by web crawlers. All these invisible documents are referred to as the *deep web*. Table 1 gives an estimation of the size of the web for in 2000 [Bergman,], 2003 [Lyman et al., 2003], and 2006 [Hawking, 2006]. Due to its invisibility, it is becoming increasingly hard to estimate the size of the deep web, which explains why no values are found for the year 2006. However, experts estimate the deep web to be over 500 times bigger than the surface web.

| Web | Year 2000 | Year 2003 | Year 2006 |
|-------------|-----------|-----------|-----------|
| Surface Web | 19 | 167 | 400 |
| Deep Web | 7500 | 91850 | - |

Table 1.1: Size of the World Wide Web in terabytes for the years 2000, 2003, and 2006.

2. A user must know the exact keywords contained in a document if she wants to have a chance to find it. As documents get indexed by keywords found inside them, this implies that a document can only be retrieved according to those keywords. Imagine for example that you are searching for holidays in Switzerland. A simple query in a search engine would be: "holidays Switzerland". Such a query will usually return thousands of results. On the other hand, all the documents that only contained the

terms *vacation* will not be retrieved as a traditional search engines ignores the fact that the words holidays and vacation are synonyms.

3. A user not only searches the web but they also browse it [Olston and Chi, 2003]. Take for example our "holidays Switzerland" query, this is a typical query made by a user who is browsing the web. As the user isn't looking for anything specific, she enters vague keywords, and then browses through the returned documents. Searching and browsing are two fundamental user behaviors on the web, but they are also fundamentally different. Search engines focus on the former behavior, but users actually spend more time browsing. Figure 1.1 shows the results of a 1995 survey⁴ that revealed that people spend more time browsing than searching. This fact is still true today. "A huge amount of time online is now spent doing the equivalent of TV channel hopping, where the consumer is surfing with an open mind actively looking for new experiences and stimulation...this is completely different to the task-orientated behaviour that search marketing closely targets." said Utarget chief executive Phil Cooper - May 2007.

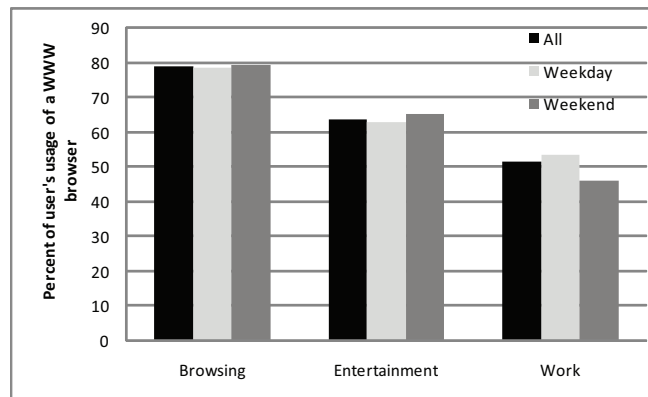


Figure 1.1: The three primary usage of a World Wide Web browser in 1995

In the middle of the 90's, many researchers saw the limitation of traditional search engines, and focused their work on helping the user on her browsing experience. *Recommender systems* were born. Recommender systems have been designed to help people browse through a collection of items based on the preferences of the user and/or others. Without no doubt, the most popular recommendation technique is *collaborative filtering*, which uses the experience of other users to recommend items. Amazon.com⁵, with its 29 million customers and several million catalog items, uses item-based collaborative filtering to recommend items to the user [Linden et al., 2003]. This technique is commonly known to end-users as "Customers who bought this item also bought these items". In practice, collaborative filtering is the most popular recommendation technique, and this is due to three main reasons. First, when sufficient preferences from the user are available, studies have shown it to have reasonably good prediction accuracy. Second, the cognitive requirement on the user is very low. Finally, it can recommend items without modeling them, as long as they have been previously rated. However, it has been argued by many authors

⁴http://www.gvu.gatech.edu/user_surveys/

⁵www.amazon.com

[Li et al., 2005, Mobasher et al., 2004, O’Sullivan et al., 2004] that collaborative filtering suffers from profound problems that will be discussed in more details in the next chapter.

Current recommendation techniques suffer from many profound problems that I believe are due to two main reasons: inadequate model of the items in the databases and incomplete user’ preferences. This dissertation presents a new recommender system called *ontology filtering* that focuses on these two problems.

1.2 Foundations in recommender systems

Before defining the problem, background knowledge about recommender systems must be set and common terms clarified.

Recommender systems are information systems that are built using a 3-tier architecture. The 3-tier paradigm is the most commonly used architecture when computer applications require a client-server model. The fundamental idea behind this model is to separate the presentation, the logic and the data storage so that they can be implemented and maintained separately. As shown in Figure 1.2, there are three layers that compose such an architecture, which are as follows:

- **The Presentation Layer:** contains the graphical user interface, GUI, that allows a person and system to interact with each other. A GUI is any computer program that allows the user to enter a query and displays information from the application layer. A typical example of a GUI is a *web browser* such as Internet Explorer or FireFox.
- **The Application Layer:** is in charge of all the logic and computation. Given a user query from the GUI, the application layer processes it by using any available data from the data layer, and then sends the results back to the GUI.
- **The Data Layer:** is responsible for storing and accessing the data required by the application layer.

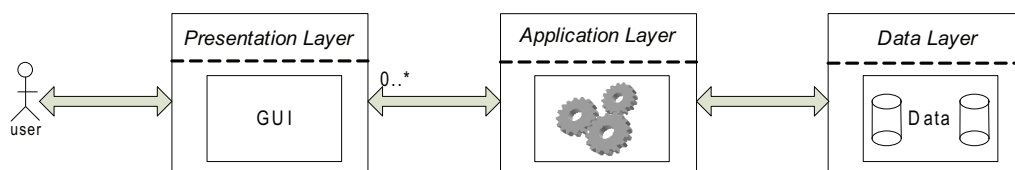


Figure 1.2: The three tier architecture

Note that a presentation layer is attached to a given user. This implies that there will be as many presentation layers as there are of users. In figure 1.2, this fact is modeled by the notation $0..*$, where $*$ means any finite number. Moreover, the layers are usually distributed such that the presentation layer is found on the user’s computer, while the application and data layers are hosted on a server somewhere in the world.

Figure 1.3 illustrates the architecture of a recommender system using the 3-tier application architecture. A recommender system should contain the following elements:

- **The Users** that are connected to the systems. Note that the terms *user* and *person* are commonly used in computer sciences, but they represent two different perspectives. Once a person is connected to the presentation layer, the system sees her as a user, which will be identified by a unique number called an identifier. Another fundamental difference is that a person is a human being like the author of this thesis, while a user is a computer representation of a person that can be deleted when not needed. This thesis focuses on the algorithms used in the application layer, which means that it has a system point of view. As a consequence, this dissertation uses the term user rather than person.

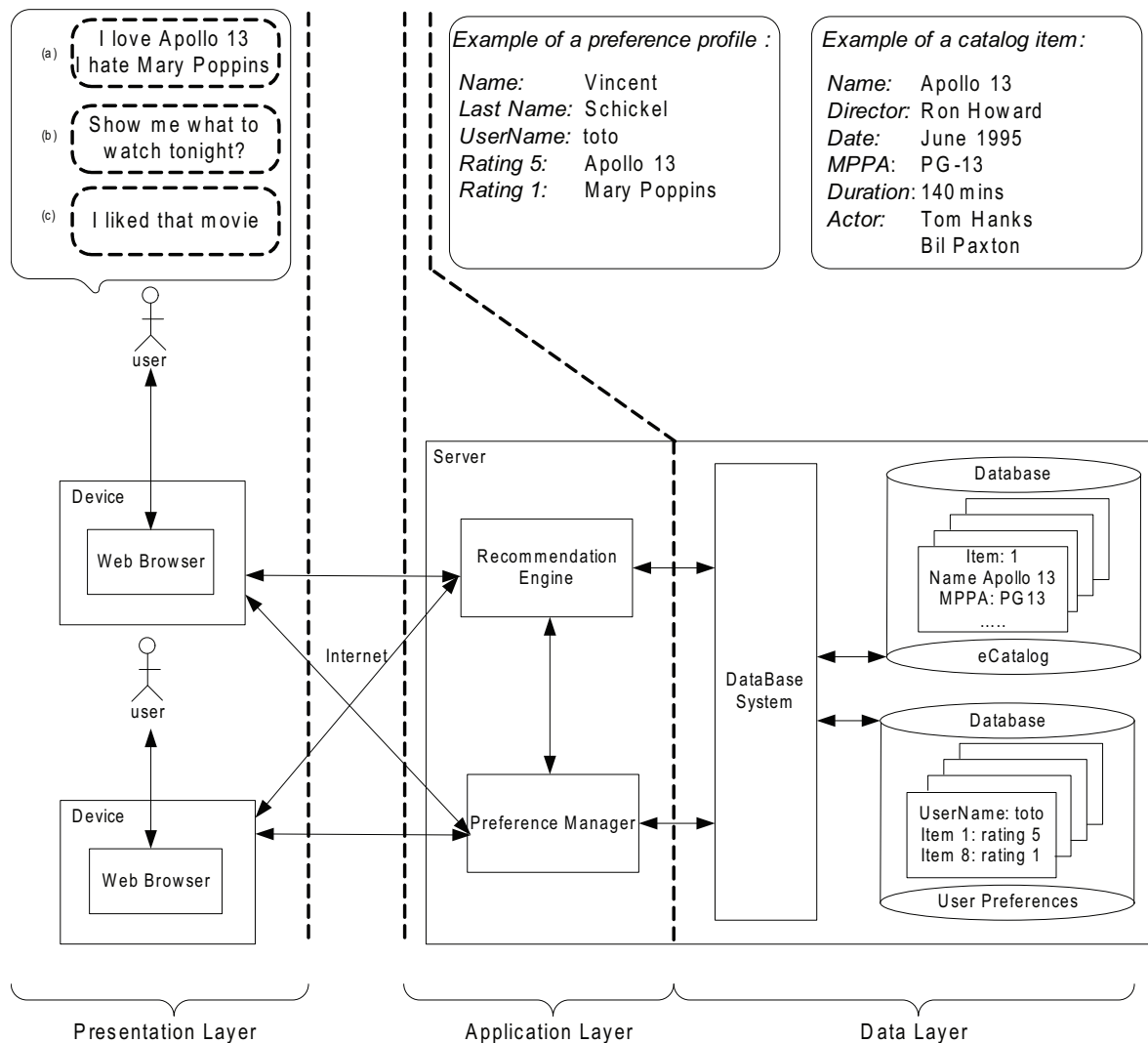


Figure 1.3: Basic component of a recommender system following the 3-tier paradigm

- **The Device's Web Browser** allows the user to interact with the application layer. The device does not necessarily need to be a computer, but it can be anything that has a web browser running on it. Mobile phones, PDAs, and notebooks are examples of such devices.

- **The Recommendation Engine** contains all the logic of the recommender system. It is this element that computes the recommendations made to users.
- **The Preference Manager** handles the *preferences* of a user. A preference models what a user likes and dislikes, while the whole set of preferences represents the *user's preference profile*. There are two ways for finding out the users' preferences: explicitly or implicitly. Explicit preferences are obtained by asking direct question to the user. For example, an explicit preference of a user could be: *I love movie Apollo 13* or *I hate Mary Popins*. On the other hand, implicit preferences are collected automatically by a computer program, without the intervention of a user. Typically, a system assumes that you liked a video if you saw its entire content. However, in practice, most recommender systems and ontology filtering use explicit preferences. As a consequence, this dissertation focuses on recommender systems that use this kind of preferences. When explicit preferences are involved, the preference manager is responsible for three fundamental tasks:
 - *Preference Elicitation* which creates the user's preference profile by asking a set of questions to the user about what she likes or dislikes.
 - *Preference Feedback* which asks the user whether or not she likes the recommendations that were proposed to her.
 - *Preference Maintenance* which allows the user to update her preferences, but also delete old preferences profiles assigned to users who do not exist anymore.
- **The Database system** is responsible for the managing the universe of discourse of the recommender system. Concretely, there are two kind of information that are handled by this element:
 - *The preference profiles* of all the users that are accessed by the preference manager.
 - *eCatalog* that represents all the catalog items a recommendation engine can recommend to users.

As shown in Figure 1.3 the database system usually stores the eCatalog and preference profiles in separate databases. To efficiently manage all this information, the database system must obey the famous *ACID* properties:

- **Atomicity** - is the ability for the database system to ensure that either all the tasks given to the database system have been performed, or none of them. A typical example is the bank transfer, where the atomicity property guarantees that an account will not be debited if the other account is not credited.
- **Consistency** - ensures that the database always remains in a stable state, before and after a query is executed. This property is very convenient when rules, known as consistency constraints, need to be implemented. Such a rule could be that no money can be debited from an account if the balance after the transaction is negative.

- **Isolation** - allows to isolate the execution of a query from an other program so that intermediate state of a query cannot be seen. In our bank transfer example, that means that a banker cannot see the money gradually move from one account to another; it is either on one account, or on the other one.
- **Durability** - is probably the most famous property, that states that once some data has been stored in the database, it cannot be undone. To finish our bank example, this property assures that once a customer has put some money on the account, it cannot be removed unless she performs a transaction.

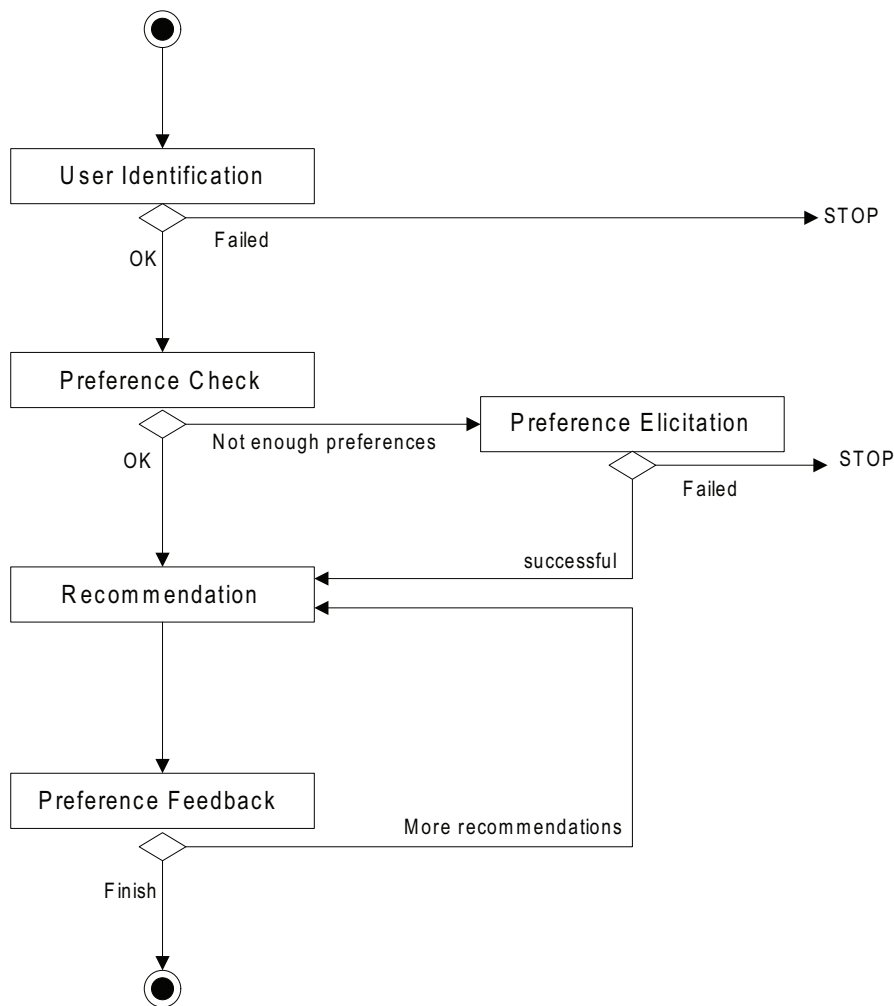


Figure 1.4: The recommendation process of a recommender system

Finally, Figure 1.4 shows the 5 common steps involved during the recommendation process of a recommender system:

1. The user starts by identifying herself to the recommender system through her device's web server. If the user cannot be identified, then the process stops. In eCommerce environment, a web page will usually follow and ask the user if she wants to register to the system.

2. The preference manager asks the database system to retrieve the user's preference profile. If the user's preference profile is valid, then the recommendation process goes directly to step 4 otherwise it goes to step 3.
3. If for some reasons the user's preference profile is invalid for making recommendations, then the preference manager needs to elicit more preferences from the user. There are mainly two reasons when this situation occurs. Either it is a new user, which means that the system cannot have any preferences; or the user has deleted too many preferences from her profile.
4. Based on the user's preference profile, the recommendation engines will generate a set of recommendations. Once the recommendations are generated, they get sent to the presentation layer for display.
5. After processing the recommendations, the user has the possibility to give some feedbacks. A possible feedback could be: *I really liked that movie*. These feedbacks are used by the preference manager to update the user's preference profile. Finally, the user has the option to stop the process or ask for some more recommendations. In the latter situation, the process return to step 4.

1.3 Problem definition and the proposed solution

This dissertation focuses on the *recommendation problem*, which is formally defined as follows.

Given a low risk context and a collection of items I , recommend N items to a user based on her preferences, where $N \subset I$.

For several reasons, this is a difficult problem. First, users usually have only limited knowledge of the content and structure of the items available in I . This seriously limits what a recommender system can elicit from the user. Second, the collection of items I can contains millions of items, but users expect answers in less than a second. Finally, this work considers low risk contexts, where users are not prepared to spend time expressing their preferences, as the risk of failure is low. Example of low risk context are renting a DVD, buying a book, listening to CD, reading jokes, going to a restaurant, buying a computer and so forth. However, bank loans, retirement plans and real estate are typical high risk context, where the approach proposed in this dissertation may not be as suitable.

To illustrate the kind of recommendation problem this dissertation tries to solve, imagine a person who wants to rent a DVD for the evening. Most people want to see a movie that they have not previously seen, which means that the recommender system needs to be able to recommend DVDs about authors they might never have heard of. DVD stores contain thousands of DVDs (my little DVD store down the road has 2'700), and online stores like Netflix have over 80'000 titles. Thus, it is unfeasible to show a list of all the available movies, and the recommendations must be produced quickly and usually within hundreds of milliseconds. Finally, as the user's life does not depend on the outcome of a

movie, she will usually not accept to rate more than 5 movies before getting the first set of recommendations.

Moreover, a good recommender system should satisfy the following properties.

- **Accurate** - the recommendations should follow the user's preferences, and only items that will be liked should be recommended to the user.
- **Scalable**- the recommender system must be able to handle millions of items, and produce recommendations within less than a second.
- **Elicitation** - the recommender system cannot ask too many questions from the user.
- **Privacy**- User's preferences are vital and owned by the user. These preferences should not be distributed, and no other user should be able to access someone else preferences.

To solve this problem, the recommendation engine and preference manager of Figure 1.3 needs to be designed and implemented. Current recommender techniques exist, but they suffer from many profound problems that I believe are due to two main reasons: incomplete users' preference profiles, and inadequate model of the eCatalog. This dissertation proposes a novel recommender system called ontology filtering that overcomes both problems. The fundamental idea is to use the characteristics and information of a data structure called an *ontology* in order to add a structure to the items we want to recommend. This structure allows to restrict the search space, and infer missing user's preferences in order to reduce the elicitation overload.

1.3.1 Potential applications for ontology filtering

There are many domains that could benefit from ontology filtering. The following examples illustrate the possible, but not exclusive, applications where ontology filtering could have a potential benefice.

Online retailer is the equivalent of downtown retailer store, but on the World Wide Web.

Popular examples of online retailers are Amazon.com⁶, Wal-Mart⁷, Ebay⁸, CVS pharmacy⁹, and so forth. In the US only, this sector grew by 25% to \$220 billion last year, which is significantly higher than the 20% prediction made in 2006¹⁰. To illustrate these numbers, Figure 1.5 shows the revenues generated by online retailers over the past five years in the US.

Forrester Research, the author of theses surveys, explained the huge levels of growth by user-friendly web sites, and the widespread broadband internet. As recommender systems help the user's browsing experience, they have the potential to further increase the revenues of online retailers by converting browsers into buyers, and increase loyalty through a customized browsing experience. Currently, only major

⁶www.amazon.com

⁷www.walmart.com

⁸www.eBay.com

⁹www.cvs.com

¹⁰2006 Annual Survey by Forrester Research for Shop.org

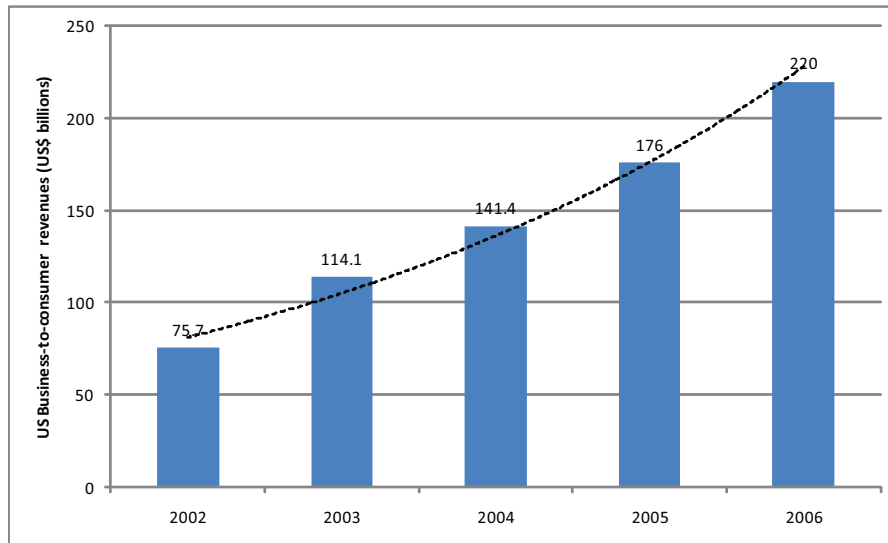


Figure 1.5: US business-to-consumer revenues in US\$ billions over the past five years. Data was obtained from the annual surveys by Forrester Research for Shop.org

online retailers such as Amazon.com, Wal-Mart, Yahoo!Music host recommender systems. These recommender systems use various versions of collaborative filtering, which have known limitations. These limitations, such as the scalability problem, limit the use of these techniques to major online retailers. Ontology filtering does not suffer from these limitations, and thus could potentially help small and medium retailers compete with the big ones.

Entertainment related activities represent the second primary usage of the web. Furthermore, pop star singers like Britney Spears and movies are always among the 10 most popular searches on the internet. MyStrands¹¹ and Pandora¹² offer tools that help the users find which song to listen to. Netflix¹³ hosts a recommendation engine that helps users living in the US to rent DVDs online. Netflix has been made famous in the recommender systems community by creating a \$1 Million dollars competition. The rule of the competition is very simple: anyone who can improve the accuracy of their recommender system by at least 10% gets the million. Beyond the fun of this competition, it reveals the need for new recommender systems.

Advertising is probably one on the biggest source of revenue of the web, and many companies' revenues deeply rely on it. For example, in 2004, Google made 96% of its revenue through online advertising. Currently, there are two methods for displaying online ads:

- The *static* method is the most popular approach, and consists of displaying a set of predefined ads somewhere on the page.

¹¹www.mystrands.com

¹²www.pandora.com

¹³www.netflix.com

- The *keyword* method is becoming increasingly popular thanks to the success of Google. Contrary to the static method where ads are predefined, the keyword approach chooses which ad to display based on the keywords it has been labeled with.

The keyword approach allows the advertisers to personalize the ads based on the user's search criteria. This is very useful as users are more likely to buy something if it is relevant for them. Take for example a user who types the following keywords in a search engine: "+computer +buy". This query tends to indicate that the user is looking to buy a computer, which means that showing ads about vacation in Switzerland will not be as useful as showing her an ad for Dell¹⁴ computers. Unfortunately, the keyword approach can only recommend ads based on a set of keywords. To make things worse, the set of keywords is usually small as the cost of an ad is correlated to the number of keywords it contains. As ontology filtering uses an ontology in its recommendation engine, it can recommend ads that have similar words like synonyms. Thus, ontology filtering has the potential to recommend more personalized ads to the user, and increase revenues for the advertisers.

1.3.2 Applications not suitable for ontology filtering

This dissertation does not claim to be able to solve all recommendation problems. The main reasons lies in the assumptions made in our model. Ontology filtering makes the assumption that users have pessimistic behaviors, and that the domain of the recommendation has a low risk factor. There are many situations where these hypotheses do not hold. The following domain are example of this situation.

Financial Industry needs to be personalize investments in order to attract customers and minimize the investment risk. However, the financial industry is a very high risk sector, where users are prepared to spend over an hour learning about the various options. In this situation, classical preference based approaches are more suitable, as they can theoretically find the optimal solution.

Medical domains are also using computer tools to diagnose the illness a patient is suffering. This is a life critical domain, where any mistake could seriously put the patient life at risk. The inference mechanism of ontology filtering uses common knowledge of an ontology to infer missing values, but illnesses are usually patient specific. For such domain, rule based systems are usually more appropriate.

Gambling is becoming increasingly popular on the web, and the anonymity of the internet allows people to easily bet on almost anything, while maintaining their privacy (to a certain extend!). By definition, gamblers like to take risks, which violates the risk pessimistic behavior of the ontology filtering domain.

¹⁴www.dell.com

1.4 Contributions

The main contributions of this thesis can be briefly summarized as follows:

1. **An ontological model** to represent and reason over the items of an eCatalog using an ontology. This ontology allows to reason over heterogenous items, and is the key element of ontology filtering's inference mechanism.
2. **An inference mechanism** that can infer missing user's preferences based on known preferences and information contained in the ontology. This algorithm allows to reduce the elicitation process, while preserving the recommendation accuracy.
3. **A similarity metric** for computing the similarity between pairs of concepts in an ontology. The inference mechanism starts the transfer of preference from the closest concept in the ontology on which the user has set some preferences. Unfortunately, current similarity metrics do not achieve high correlation with users' judgements. By modeling the pessimistic behavior of a user, the inference mechanism of ontology filtering is extended to build a more robust similar function called *OSS*.
4. **An ontology learning algorithm** that can automatically construct a set of ontologies if none are available. This algorithm uses existing preference profiles and unsupervised learning algorithms to generate a set of 15 different ontologies. An algorithm for selecting which ontology to use based on the user's preferences is also presented. Experiments show that use of personalized ontology performs better than using the same ontology for every user.
5. **Ontology Filtering** proposes a new recommender technique that satisfies the 4 main properties of a recommender system defined in Section 1.3. Furthermore, experiments on real data sets have shown significant improvement of the recommendation accuracy.
6. **Validation** of the main contributions are provided on real data sets. Formally, the recommendation engine is validated on the Jester and MovieLens data set, while the similarity metric is tested on WordNet and the GeneOntology.

1.5 Overview of the dissertation

As shown in Figure 1.6, this dissertation is composed of nine chapters that can be summarized as follows:

Chapter 1: Introduction starts by giving the motivation of this work, and emphasizes the need for new recommendation systems. Before defining the problem, background knowledge about recommender systems is set, and common terms are clarified. The problem that this dissertation is trying to solve is clearly defined, along with some example applications that could benefit from of ontology filtering.

Chapter 2: Background looks at the state of the art in recommender systems. Collaborative filtering and the preference based approach are used as comparison as there are the most popular techniques used by online retailers. Finally, this chapter highlights the differences of ontology filtering with existing techniques. This analysis allows to position ontology filtering as a knowledge based recommender systems.

Chapter 3: Modeling eCatalogs with Ontologies shows how to use an ontology to model the eCatalog. This chapter also shows that it is possible to extract the information contained in an ontology by considering the number of descendants of a concept.

Chapter 4: The User's Preference Profile defines the user's preference profile and explains why an ontology is not used for such model.

Chapter 5: Inferring Missing User Preferences defines the inference mechanism that is used to predict the unknown preferences of a user. Following these mechanism, this chapter proposes to extend the inference idea to build a similarity function that can compute the similarity between pairs of concepts.

Chapter 6: Learning the ontologies acknowledges the fact that assuming the existence of ontology is not realistic in real life. To overcome this limitation, three algorithms are introduced. The first one shows that it is possible to learn these ontologies by using unsupervised clustering algorithm, while the second proposes to select an ontology based on the user's preferences rather than using the same ontology for all the users. Finally, a third algorithm is introduced that can further increase the prediction accuracy by building more complex structures that allow concept to have more than one parent.

Chapter 7: Ontology Filtering describes the architecture of the proposed recommender system. Each component is defined in details in chapter 4,5 and 6, but this chapter allows the reader to better understand how they interact with each others.

Chapter 8: Experimental Results is in fact divided in three main sections. The first section validates the inference model on the WordNet and GeneOntology, while the other two validate the ontology filtering recommender system on the MovieLens and Jester data sets. It is important to point out that all these experiments use real user data, not simulated one.

Chapter 9: Conclusions provides a brief summary of these dissertation and its contributions. Limitations and future research directions are also discussed.

Bibliography contains all the references used in this dissertation.

Appendix A gives a classification of the different recommendation techniques.

Appendix B contains the proof that the similarity function derived from the inference mechanism is transitive.

Appendix C illustrates the ontology filtering approach on a joke example.

Appendix D explains in details how the WordNet and the GeneOntology are structured.

Appendix E contains extended experimental results for Experiments I, III, and VIII.

To allow ontology filtering to reason over incomplete preference profiles, this dissertation defines nine fundamental algorithms. A brief description of these algorithms can be found in the *List of Algorithms* section, which is located at the beginning of this document. Figure 7.2 at page 111 shows how each algorithm interacts, and in which order that are executed.

Following EPFL's regulations, this thesis begins with an abstract in both English and in one of the three official languages (i.e. French). Similarly, the thesis ends with the curriculum vitae of the author, along with its publications written during his PdD.

1.6 Intellectual property protection

Some of the ideas and algorithms presented in this dissertation are under the protection of a provisional patent (number 60/819,20), while the US patent number 11/775,106 is pending. As a consequence, the content of this dissertation can only be used for educational or research purposes. Any other usage could violate intellectual property law, and the authors may be prosecuted.

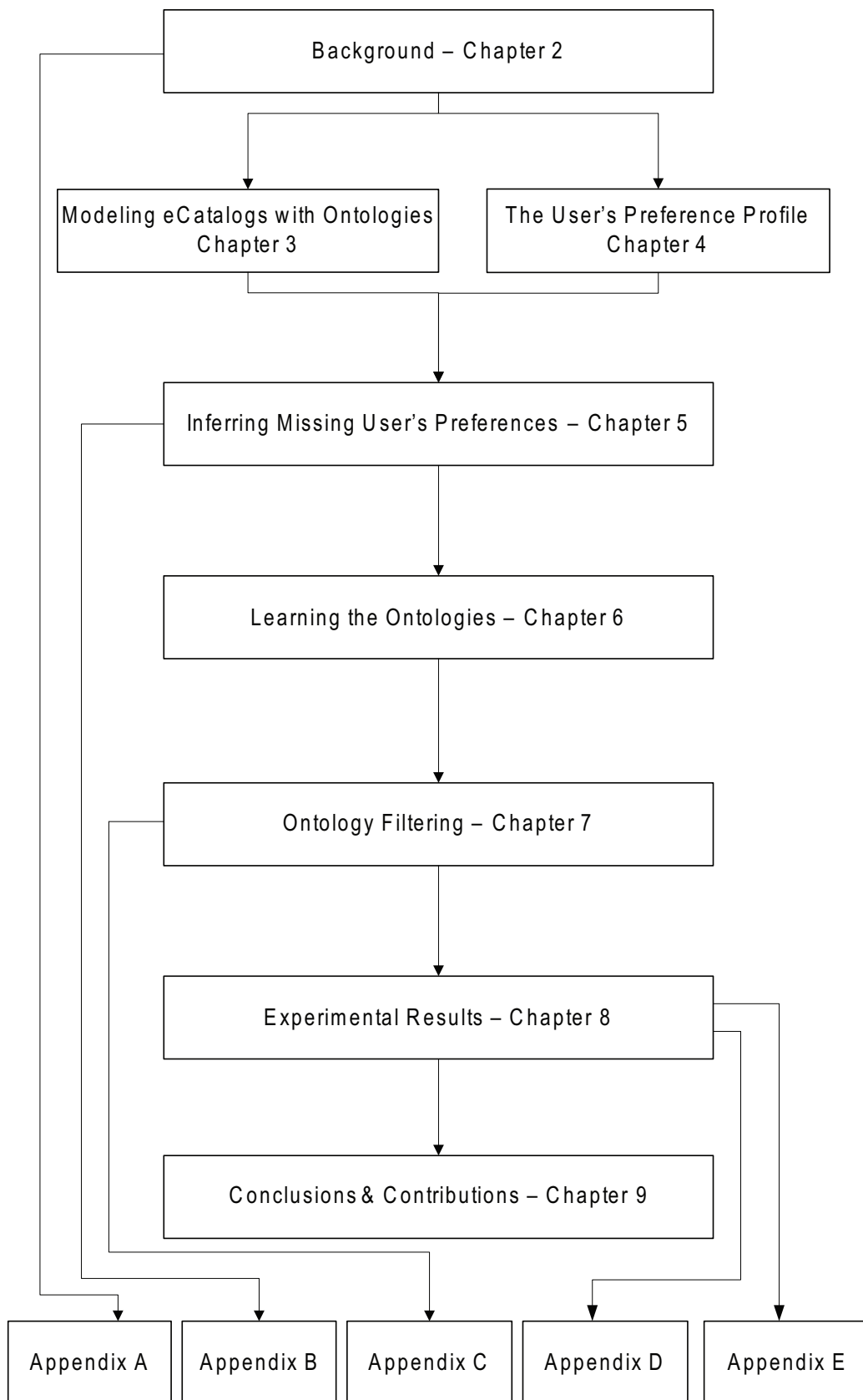


Figure 1.6: Layout and dependencies between each chapters in this dissertation

Chapter 2

Background

Since the rise of the web, recommender systems have become extremely popular among the research community and online retailers. This chapter explains the main techniques used to build recommender systems, and illustrates their applications with concrete examples. Among these techniques, collaborative filtering and the preference based approach will be looked at in more details, as they are widespread among the big online retailers. This state of the art summary allows to position ontology filtering among existing techniques.

2.1 Evaluating recommender systems

Before introducing in detail the various recommender techniques, it is important to clarify what a recommender system should do, and how to evaluate it.

A typical recommender system makes recommendations by returning a set of N items that it thinks the user will like best. This is typically called the *top- N recommendation strategy*. The first question that arises is how to evaluate these recommendations?

There are two important aspects to consider when evaluating the recommendations made by a recommender system:

- *Accuracy* measures how much a user likes the recommendations that are made to her. Ideally, a good recommender system should have all of its recommendation liked by the user (i.e 100% accuracy).
- *Novelty* looks at how non-obvious the recommendations are. To understand this aspect, consider a recommender system that recommends movies to users. Further imagine a user *Alex* that said that he liked the last Harry Potter movie - The Order of the Phoenix. An obvious recommendation would be to recommend the last four Harry Potter movies. Even if these recommendations have a high chance of being accurate, there are potentially useless as *Alex* has probably seen these movies before.

2.1.1 Measuring the accuracy

Many metrics have been proposed for evaluating the accuracy of the recommendations made by recommender systems. The most famous metric is the *Mean Absolute Error*, (MAE, [Sarwar et al., 2001]), which evaluates the accuracy by measuring the mean average deviation between the expected ratings and the true ratings. Formally, the MAE of a set composed of N recommendations is computed as follows:

$$MAE = \frac{\sum_{i=1}^N |(r_i - \tilde{r}_i)|}{N}, \quad (2.1)$$

where r_i is the real rating assigned by the user, and \tilde{r}_i is the rating estimated by the recommender system. Later on, [Herlocker et al., 2004] argued that this metric was not the most appropriate when considering rated items for users, as a user is usually interested to know if she is going to like the items or not. In our movie example, imagine the situation where the recommender system recommends the *The Lion King* movie to Alex. Alex does not care if the estimated rating is 4.5 or even 5.5, but just wants the recommendation to be pertinent.

As a consequence, [Herlocker et al., 2004] proposed to use the classical information retrieval metrics: *precision* and *recall*. Precision measures the proportion of relevant items in the set of recommendations returned by the recommender system, while recall is defined as the number of relevant items retrieved over all relevant items in the database. Figure 2.1 illustrates this two metrics, where the top-N ellipse is the top-N recommendation set, and the liked items ellipse represents all the items in the database that are liked by the user. Note that both precision and recall metrics focus at the intersection of these two sets. However, precision divides the number of items in the intersection over the number of items in the top-N recommendations, while recall divides it over all the possible items in the database that are liked by the user.

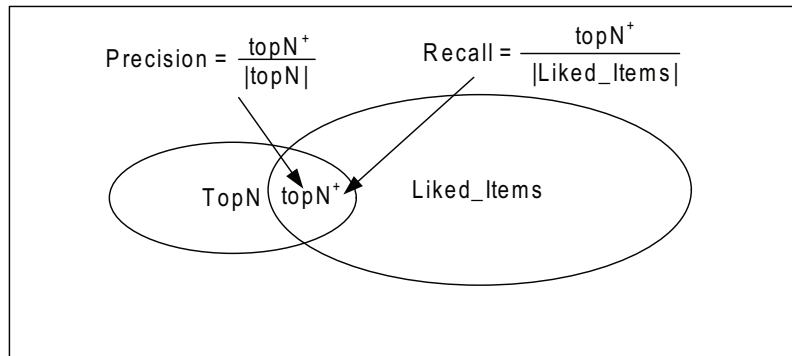


Figure 2.1: Illustration of the precision and recall metrics

Formally, given a recommendation set $topN$, precision is defined as the ratio of relevant items $topN^+$ to the total number of items shown in $topN$, while recall is defined as the ratio of relevant items to the total number of relevant items available in the database, $Liked_Items$.

$$Precision = \frac{topN^+}{|topN|}; Recall = \frac{topN^+}{|Liked_Items|} \quad (2.2)$$

There are two challenges when using precision and recall for evaluating the accuracy of recommender systems. First, precision and recall need to be considered as a whole to fully evaluate the accuracy of a recommendation. Second, it has been observed in many applications that precision and recall are in fact inversely related. Thus, it is necessary to use a metric that can combine both precision and recall. As a consequence, and following the results in [Herlocker et al., 2004], the *F1 metric* evaluates the accuracy of a recommender system by combining precision and recall into a harmonic mean. Formally, the F1 metric is defined as follows:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}. \quad (2.3)$$

2.1.2 Measuring non-obvious recommendations

Novelty and serendipity are two metrics that measure non-obvious recommendations, where the former concentrates on new items with known features, while the latter measures surprisingly new items, even in their features. By definition, serendipity implies novelty, but unfortunately, serendipity is impossible to compute without knowing exactly the user's preferences - which are unknown. Measuring the novelty of the recommendations remains a hard problem, and usually requires asking the user specific questions about the features liked by the users.

In this dissertation, an algorithm is proposed that learn a set of taxonomies based on the experience of previous users. One property of these learnt taxonomies is that the features making an item are implicitly hidden in the edges. Thus, this makes it extremely hard to evaluate the novelty without asking the user. As a consequence, this thesis will not focus on this aspect.

In the early stage of this work, ontologies were manually created with edges representing explicit features. In this case, novelty is slightly easier to model. Thus, this dissertation proposed to use the novelty metric defined by Equation 2.4, which measures the number of correct recommendations made by algorithm a that are not present in the recommendations made by the popularity strategy. The popularity strategy simply returns the N items that are the most popular among the users in the database.

$$Novelty(topN_a) = \frac{(|topN_a^+| - |topN_a^+ \cap topN_{popularity}|)}{N} \quad (2.4)$$

where $topN_a$ are the top- N recommendations made by the algorithm a , $topN_a^+$ are the correct recommendations contained in $topN_a$ (i.e. the items liked by the user), and $|topN_a^+ \cap topN_{popularity}|$ is the correct recommendations made by algorithm a that are also present in the recommendation set of the popularity approach.

2.2 Overview of recommender systems techniques

Recommender Systems can be categorized in various ways that depend on a number of criteria such as the input data, the algorithms used, the feedbacks, and so forth. Over the years, researchers have come with different ways to categorize recommender systems that reflect the issues of the time.

[Resnick and Varian, 1997] categorized recommender systems from both a technical design space and from a domain space. The technical design space is made of the five dimensions: the contents of the preferences, explicit vs implicit preferences, whether or not the preferences are anonymous, the algorithm used to aggregate these preferences, and the way the aggregated preferences are used. Note that the authors used the term recommendation to denote the preference of a user. The domain space analysis is subdivided in two parts. First, various characteristics of the recommended items such as the type and lifetime of the items are evaluated. Then, it is the characteristics of the users that are evaluated based on whether they participate in the recommendation process, or if they are simple consumers of the system.

[Schaffer et al., 1999] focused the classification of recommender systems on three technical design aspects. The first aspect that was studied was the recommendation interface used for displaying the results, while the second looked at which recommendation algorithm was used. These two issues were also studied by [Resnick and Varian, 1997] and correspond respectively to their 4th and 5th technical design aspects. The third aspect looks at the actions that a user must perform in order to get some recommendations.

[Terveen and Hill, 2001] identified four main issues in recommender systems, which were then used to categorize the various recommender techniques. These issues can be briefly summarized as follows:

Preferences are key elements, as recommendations are based upon them. As a recommender system must obtain preferences, this raises some questions such as: Whose preferences are used? How are preferences obtained? What incentives are there for people to offer preferences? How are preferences represented?

Roles & Communication of the users and recommender systems in order to obtain some recommendations. Do all people play the same role? Are role fixed or do they evolve over time? Is the recommender role filled by a computational system or by person? If information about preference providers is revealed, are any measures taken to safeguard privacy?

Algorithms for computing the recommendations. How are recommendations computed? Do you combine the preferences of the neighbors, or use a weighted average?

Human-Computer Interaction focuses on how to present the results to the user. Do we use ordered lists, or more complicated visual annotations?

[Burke, 2002] did not focus on the recommendation interface, nor the issues associated to recommender systems. Instead, the author studied the available data that the system has before the recommendation process begins, the information that a user must communicate to the system in order to generate a recommendation, and the algorithms that combines the background and input data to arrive to a recommendation. Given these criteria and assumptions, Burke distinguished the following 5 recommendation techniques:

Collaborative Filtering uses previous ratings of a given community of users, and extrapolates these ratings based on the ratings provided by the user.

Demographic is very similar to Collaborative techniques. Demographic techniques categorize users based on their personal attributes and make recommendations based on demographic classes. Thus, it identifies users that are demographically similar to the user to whom we want to do the recommendation, and extrapolates from their ratings to get the recommendations.

Content-based uses the items' features as background data. Given some user's ratings on a set of items, the system builds a classifier for that user, and then uses it to recommend items. This is very different from demographic and collaborative techniques that use the ratings of a community of users.

Utility-based also uses the items' feature as background data. Contrary to the content based approach that uses ratings as preferences, the utility model requires the user to state utility functions over the attributes (features) of the items. These utility functions are then applied on all the items to be recommended, and the items with the highest valuation are selected for the recommendation.

Knowledge-based is a technique that recommends items based on the user's need. In this model, the features of the items and knowledge of how these items meet a user's needs are used as background data. Given a description of the user's needs and interests, the system infers a match between the items and the user's needs. Note that [Schaffer et al., 1999] call knowledge-based recommendation the Editor's choice method.

Burke also looked at various hybrid recommender systems. Hybrid systems combine two or more recommendation techniques to gain better performance with fewer drawbacks of any individual one. Most commonly, collaborative filtering is combined with some content based technique in an attempt to avoid collaborative filtering's problems.

[Adomavicius and Tuzhilin, 2005] simplified previous classifications, and classified recommender systems into three main categories: collaborative, content based, and hybrid recommendations. The authors briefly mentioned knowledge-based technique as a way to improve hybrid recommendation systems, but argued that there are used in very limited domain, as the approach requires some form of knowledge acquisition.

Unfortunately, this brief survey shows that authors use different terms to denote the same technology, while others use the same terms to denote different technologies. Take for example the recommender technique that works on the item's content, [Schaffer et al., 1999] call this technology *item-to-item correlation*, while [Burke, 2002] uses the term *content-based recommendation*. Moreover, [Schaffer et al., 1999] also use the term item-to-item correlation to denote the item-based collaborative filtering used by Amazon.com, which does not use the description of an item. The explanations for this mismatch lies in the definition given by Schafer et al. at the item-to-item correlation - "Item-to-item correlation recommender systems recommend products to customers based on a small set of products the customers have expressed interest in". As user's preferences are expressed over items on both item-based collaborative filtering and content based, [Schaffer et al., 1999] used the same terminology to define them.

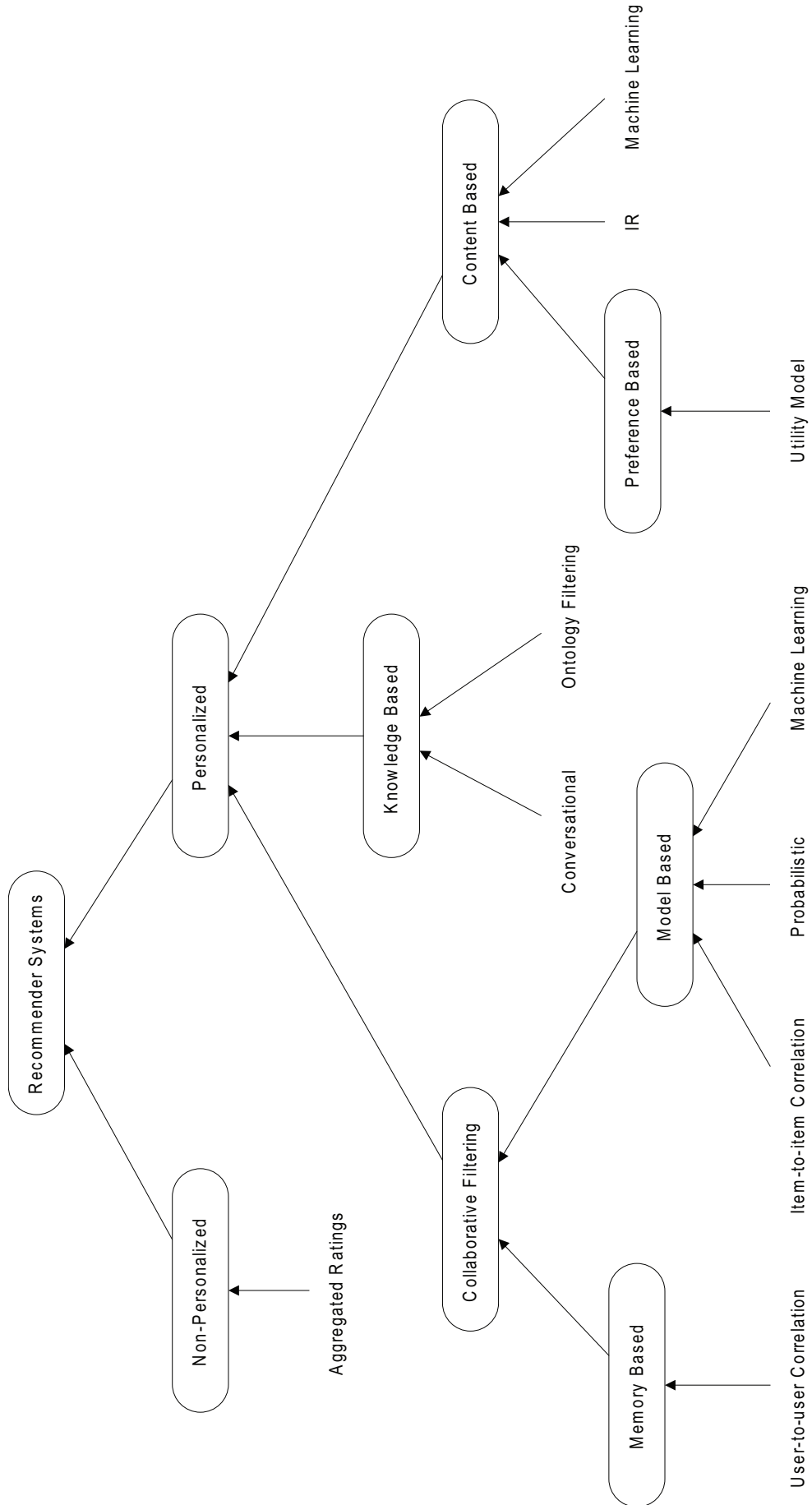


Figure 2.2: Main techniques used in recommender systems

These ambiguities in the classification terms are a barrier in understanding prior arts, and makes it difficult to position the ontology filtering approach. Note that most recommender systems use the user's preferences to make a *personalized* recommendation. This allows them to personalize each recommendation made to users. However, non-personalized recommender systems are independent from the user's preferences, which means that all the users get the same recommendations. Figure 2.2 clarifies the major recommendation techniques used today and their different relations. The reader can find a summary of the classification terms of the cited authors and their correspondence with this dissertation in Table A.2.

2.2.1 Non-personalized recommender systems

Non-Personalized recommender systems are very basic recommender systems that do not use the user's preferences to make a recommendation. Instead, an item is recommended based on the experience of previous users, which means that it is independent of the user (i.e. all the users get the same recommendations). The main drawback of non-personalized techniques is obvious - it is not personalized. On the other hand, it has many advantages. First, it is extremely fast as the recommendations can be made off-line. Second, it is easy to implement and does not require too much computational resources. Finally, users feel more confident about the recommendations, as they are very easy to understand.



Figure 2.3: This snapshot illustrates the non-personalized recommender system of Moviefinder.com, where a news item is recommended by average ratings of previous readers.

Today, the most popular technique on the web is the *aggregated ratings* technique. This technique recommends an item based on the aggregation of previous users' ratings. It works in two basic steps. First, users' ratings for given items are collected, and aggregated

using a mathematical function (such as the minimum, average, sum, and so forth). Then, items are sorted based on the aggregated ratings, and the recommended items are the ones optimizing the mathematical function.

In practice, the *average* function is currently the most popular on the web. For instance, Moviefinder.com¹ recommends news items by averaging ratings assigned by previous readers. Any item can be given a 5 star rating by a user, where the number of stars is proportional to the reader liking the item. Figure 2.3 illustrates this recommendation technique on the Top Stories section. The top story on June 10th, 2007 was: *Paris Won't Appeal Jail Sentence*, and was recommended as all the users gave this story the maximum rating of 5 stars. Notice that the Most Popular News section can also be sorted based on the number of comments, the amount of times it was emailed, the number of times it was read, or by the number of comments people have left.

The Web 2.0 is also starting to use the non-personalized recommender technique to recommend items to a community of user. Take for example Digg.com², it recommends news items based on the maximum number of *diggs* a news article gets. When a user likes an item, she digs it by pressing a button. It is this simple counter which is called digg. Figure 2.4 shows Digg.com's top news on June 10,2007; where the top news has a digg value of 4987.

The screenshot shows the Digg.com interface. At the top, there are tabs for 'All News', 'Popular Stories', and 'Upcoming Stories (5,177)'. Below the tabs, there are filters for 'Newly Popular', 'Top in 24 Hours', '7 Days', '30 Days', and '365 Days'. The main content area displays a list of news items, each with a 'digg' count and a 'digg it' button. The top item is 'Poor Pluto [picture]' with 4987 diggs, submitted by 'sprice' and made popular 20 hours 47 min ago. The second item is 'McDonald's Wants To Manipulate The English Language' with 3234 diggs, submitted by 'tropicari' and made popular 15 hours 47 min ago. The third item is 'Average White Kid Gets Owned by his Parents over World of Warcraft !' with 2769 diggs, submitted by 'tropicari' and made popular 13 hours 47 min ago. To the right of the main content, there is a 'What's Digg?' section with a 'Join' button and a 'Learn More...' link. Below that is a 'Top 10 in All Topics' section with a list of top stories and their digg counts.

Figure 2.4: This snapshot illustrates the non-personalized recommender system of Digg.com, where a news item is recommended by the maximum number of diggs assigned by previous readers.

2.2.2 Personalized recommender systems

A personalized recommender system recommends items to a user based on her preference profile. The user's preference profile can be built from either implicit or explicit preferences, or a combination of both. Once the profile has been built, the recommender system applies one or more recommendation techniques to generate recommendations to the user. Figure 2.2 identifies three different personalized recommendation techniques:

¹<http://www.moviefinder.com/news/index.jsp>

²<http://www.digg.com/news/popular/24hours>

1. *Content-based approaches* make recommendations based on items *similar* to the ones a user has liked in the past. Two items are similar if some of their content or attributes overlap.
2. *Collaborative filtering* does not focus on the item itself, but recommend items that people with similar tastes and preferences liked in the past;
3. *Knowledge-based approaches* use both knowledge about the users' preferences and the items.

Note that these techniques can then be combined to create hybrid recommender techniques. This dissertation do not focus on this aspect, but a good survey can be found in [Burke, 2002].

Explicit preferences

Explicit preferences are the most common form of preferences in recommender systems. These preferences are collected by explicitly asking questions to the user. This is sometimes called *explicit profiling*[O'Sullivan et al., 2003]. There are currently four approaches to elicit explicit preferences[Smyth and McGinty, 2003]:

1. *Value elicitation* was the most common elicitation approach in the 1990's. Typically, a user is asked to give values to specific features modeling the items to be recommended. Figure 2.5 illustrates the value elicitation approach used by NotebookReview.com³, which recommends notebooks to end-users. The notebooks are described by the features manufacturer, model series, processor type, screen size and its price. Note that in this example, the user can state discrete values (Processor Type=Intel Core Duo), and a limited range of continuous possibilities (minimum price = \$0 and maximum price = \$999).

The image shows a web form titled "Laptop Search". It contains several input fields:

- Manufacturer:** A dropdown menu with "[Any]" selected.
- Model Series:** A dropdown menu with "[Any]" selected.
- Processor Type:** A dropdown menu with "[Any]" selected and open, showing a list of options: Intel Pentium M, Intel Core Duo, Intel Core 2 Duo, AMD Athlon, AMD Turion 64, and Intel Celeron.
- Screen Size:** A dropdown menu with "[Any]" selected.
- Price:** Two input fields labeled "Min" and "Max". The "Min" field contains "0" and the "Max" field contains "9999".

 At the bottom of the form is a button labeled "SEARCH LAPTOPS".

Figure 2.5: A snapshot of value elicitation from Notebookreview.com

³<http://www.notebookreview.com/>

Value elicitation allows the system to precisely model the user's preferences, and thus increase the quality of the recommendations. However, it has two major drawbacks that have seriously limited its use in today's recommender systems. First, the user must have a good level of expertise and understand the item's features. Unfortunately, this assumption rarely holds in eCommerce systems. Consider our notebook example, most people do not know the difference between an Intel Pentium M or an AMD Turion 64, or any other computer specific features as a matter of fact. As a consequence, most people will usually select only the price criteria, which greatly limits what a recommendation system can offer. Second, the cost of thinking, also known as the *cognitive cost*, is very high as the user must carefully think about the attributes modeling the alternatives.

2. *Rating based* is becoming the most popular way of asking the user to give explicit preferences. Here, a user simply needs to rate a given set of items on a predefined rating scale. In most cases, the rating scale ranges from 1 to 5, where 1 means really disliking the item and 5 is loving it. For example, Figure 2.6 shows how a user on Amazon.com rates 4 out of 5 a book he owns.

The Rating based elicitation approach does not suffer from the same drawbacks as value elicitation. Moreover, the cognitive load on the user is very low. This explains why so many web sites such as YouTube⁴, eBay⁵, Amazon.com are using this technique. However, one preference obtained by the rating based technique is usually not as precise as one obtained by value elicitation. This is because the systems does not know exactly why a user liked the item. For example, when a reader rates a book 4 out of 5 as shown in Figure 2.6, Amazon.com does not know if the reader liked the book because of the story plot, the author, or any other features. As a consequence, rating based approach usually requires the user to rate a lot of items in order for the recommender system to build accurate preference profiles. In collaborative filtering, this is known as the *cold start* or ramp-up problem.

Rate this item to improve your recommendations



Figure 2.6: A snapshot of rating based preference elicitation from Amazon.com.

3. *Critiquing* was introduced by [Burke et al., 1997] to tackle the problems faced by the value elicitation approach. Instead of asking the user specific questions about features, the user is shown a suggestion and asked to critique some of its features.

Unlike value elicitation, critiquing requires less expertise and less cognitive reasoning from the user. Figure 2.7 shows the critiquing process on the restaurant recommender system Entree. The user is presented a restaurant, Yashi's Cafe, and she has the possibility to critique it by looking for a cheaper meal (button *less \$\$*) or a nicer

⁴<http://www.youtube.com/>

⁵www.ebay.com

restaurant (button *nicer*). Unfortunately, this approach has two drawbacks. First, a knowledge based system must be created for assigning features to critique actions. For example, a rule must be created for assigning the feature price to the critique *less\$\$*. Second, the selection of which suggestions to show to the user is fundamental in order to motivate them to critique it. [Viappiani et al., 2006] focused on this problem by generating suggestions that will be optimal when one or more additional preferences are stated.



Figure 2.7: An example of the critiquing approach from the recommender system Entree.

4. *Preference-Based* is the simplest form of preference elicitation from the user cognitive aspect [McGinty and Smyth, 2002]. The system presents a set of items, and the user states her preferences by simply selecting one of them (*I prefer Golden Eye*). Then, the system extracts the features of interest associated to the selected item and attempt to learn detailed feature preferences. The extracted features are used to refine the search query, and the process reiterates until the user finds a solution. Initial experiments show significant reduction in preference elicitation, and the number of items seen by the user compared to showing similar items. It is also very easy to implement and can be run with a very limited user interface. Thus, it makes this approach very suitable in mobile environment, where the graphical interface is limited.

As argued by [Smyth and McGinty, 2003], it is very hard to define strict rules to govern the choice of the elicitation technique. In low risk domains, eliciting preferences is expensive as a user is usually not willing to spend much effort expressing her preferences. Thus, value elicitation and critiquing based approaches are unlikely to be useful, and should be avoided in favor of rating based or preference based approaches. Value critiquing should only be considered in medium to high risk domains as the user must have a good knowledge about the domain, and the elicitation will require a high cognitive load. As critiquing requires only little domain knowledge and cognitive loads, it can be used when the other three are not appropriate.

Implicit preferences

Contrary to explicit profiling that asks direct question to the user, *implicit profiling* observes the user's behaviors known as *implicit interest indicators* [Claypool et al., 2001]. These

interest indicators are then used to infer the user's ratings that become implicit preferences for recommender systems.

Implicit preferences have attracted less attention from the research community, but initial studies showed that some user behaviors can successfully reflect the users' intentions and interests [Hill and Terveen, 1996, Konstan et al., 1997]. [Claypool et al., 2001] studied four implicit interest indicators, and found that the time spent reading a web page and scrolling it are good indicators of interest. Moreover, their study showed that these two indicators are linearly proportional to explicit ratings. However, mouse clicks and mouse movements were not a good sign of interest, even though they showed some sign of correlation.

Today, implicit preferences are becoming increasingly popular, and five implicit profiling techniques seem to dominate.

Viewing time technique measures the amount of time a user spends interacting with an item [Konstan et al., 1997, Claypool et al., 2001, Parsons et al., 2004].

Record, playback and browsing in the digital TV domain showed to be a good alternative to explicit ratings [O'Sullivan et al., 2003].

Web usage mining is becoming increasingly popular by online retailers. This technique measures three distinct shopping behaviors [Cho et al., 2005]:

- *Click-Through*: measures the click on and the view of the web page associated to a product.
- *Basket placement*: count the number of placement of products in the user's shopping basket.
- *Purchase*: is the most reliable information and looks at the purchased products.

Word-of-mouth or user's social networking analyzes the group of friends of a user to discover her interests[Liu and Maes, 2005].

Implicit profiling has the main advantage of building the preference profile without disturbing the user's browsing experience. Although an implicit rating is likely to be less accurate than an explicit one [Claypool et al., 2001], implicit profiles are usually as accurate as explicit ones, as many more preferences are gathered. However, implicit profiling has three major drawbacks. First, it usually requires the user to install a third party software to measure the user's behavior. For example, [Claypool et al., 2001] used a specially designed web browser that captures the user's action while browsing. Scalability is also a problem as a lot of interaction has to be monitored, processed and stored. Finally, but not the least, the privacy violation is becoming an increasing concern. Furthermore, users are getting more and more concerned about having their preferences collected without their approval. To make things worse, companies who collect these preferences tend to sell them to third parties in order to increase their revenues. Note that explicit preferences can also be sold to third parties. For the user, this usually means an increase of unwanted emails. Unfortunately, the consequences for the user can be much more severe.

In 2002, Wang Xiaoning was arrested⁶ after he used Yahoo sites to distribute articles calling for democratic reform in China, and is now serving a 10 year sentence in Chinese jail. Yahoo has admitted giving the search queries that lead to this arrest, but claimed that "companies doing business in China must comply with Chinese law." Yahoo reiterated this preference give away policy to the Chinese government, which lead to the arrest and imprisonment of journalist Shi Tao⁷. As a response, the World Organization for Human Rights USA⁸ has filed a law suit against Yahoo for complicity in acts of torture and human rights abuses in China. Google is also facing legal issues⁹ with the European Union over the use of private data in Google's mail application Gmail. In May 2007, the European Union has announced that it is probing Google on privacy grounds. The outcomes of these lawsuits will most probably limit the use of implicit preferences in the future.

2.3 Content-based approaches

The *content-based* approach is a personalized recommender system that makes recommendations based on items similar to the one a user has liked in the past. The main difficulty in this approach is to find similar items to the ones the user has liked. The technique used to find these similar items depends on the structure of the items themselves, which can be categorized as follows:

- *Structured* items are described by some data that follows a well defined model. This type of data is usually stored in a database, where the data follows some kind of entity-relationship¹⁰ model. Formally, items are described by a set of attributes such as the creation date, name, creator, price and so forth. These attributes restrict the possible range of values an item can have, which create the item's model.
- To the opposite, *Unstructured* items are any items that are described by plain textual information.
- *Semi-structured* items are in between structured and unstructured data. An email message is a perfect example of semistructured item in that it has well defined header fields such as the sender, and an unstructured text body.

Figure 2.8 illustrates these different structures on the Apollo 13 movie. The structured model represents the movie by a set of attributes-value pairs such as title=Appolo 13, duration=140mins, and so forth. The semi-structured model uses the web page¹¹ describing the movie. The web page is composed of well defined fields such as the title, but the plot summary is unstructured. Finally, Apollo 13 can be described only by its plot summary, which is totally unstructured.

⁶<http://www.technewsworld.com/story/57011.html>

⁷<http://www.cnn.com/2007/TECH/internet/06/11/yahoo.china.ap/index.html>

⁸<http://www.humanrightsusa.org/>

⁹<http://searchengineland.com/070612-041042.php>

¹⁰http://en.wikipedia.org/wiki/Entity-relationship_model

¹¹<http://www.imdb.com/title/tt0112384/>

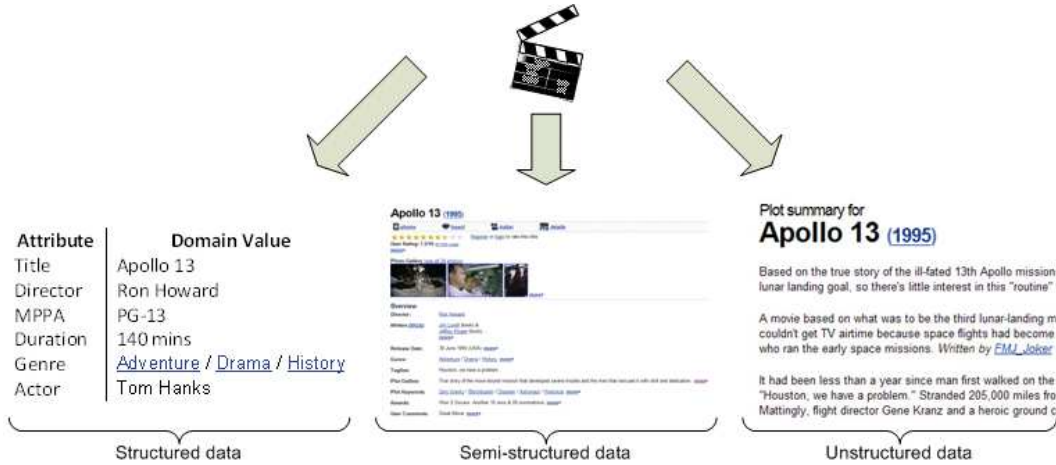


Figure 2.8: Three different views of the Apollo 13 movie: *structured*, *semi-structured*, and *unstructured* model.

When items can be described by attributes, then a *preference based approach* can be used (see Section 2.3.3). However, when dealing with unstructured or semi-structured data, the preference based becomes unsuitable and more general approaches are required. Two approaches are very popular in this situation: the *information retrieval* and the *machine learning*.

2.3.1 Information retrieval approach

Information retrieval techniques, IR, are at the root of the content-based approach, especially as they share similar objectives [Belkin and Croft, 1992].

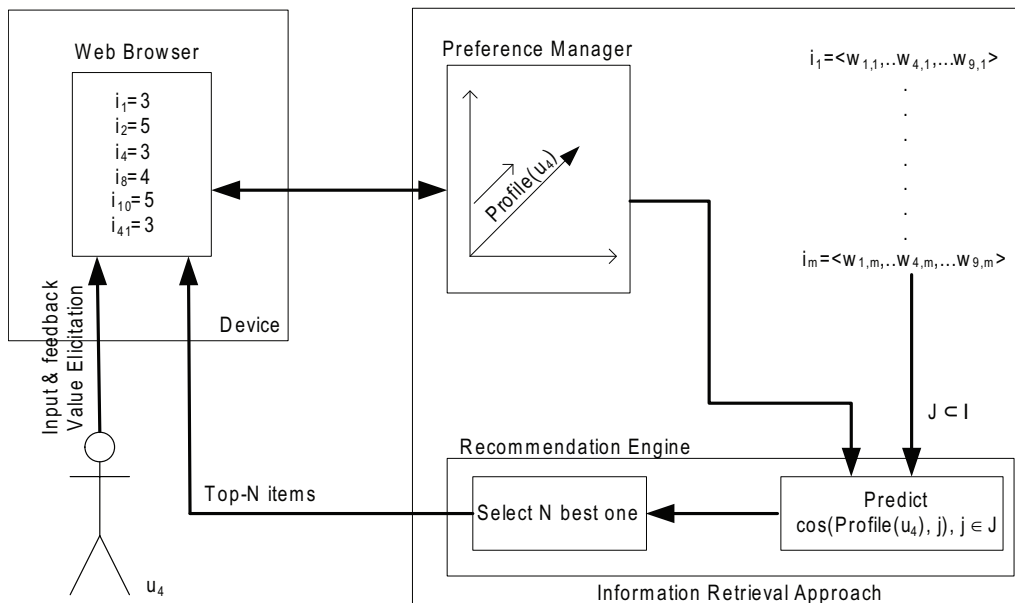


Figure 2.9: The information retrieval process.

Usually, an IR system is considered to have the function of leading the user to those documents that will best satisfy her needs for information; while a recommender system filters out items that do not meet the user's preference profile. As the content of items are usually unstructured or semi-structured, they can be seen as documents.

Figure 2.9 illustrates the information retrieval approach, which is composed of three phases.

1. *Item representation* is required to convert the unstructured data of the item into something a computer can work with. Traditionally, IR systems model documents using a vector of keywords which is known as the document vector model. In recommender systems, the set of keywords are predefined beforehand, or are learnt on a training set. For example, the Fab system [Balabanovic and Shoham, 1997] that recommends web pages to the user, represents web page content with a set composed of the 100 most popular words. Similarly, [Pazzani and Billsus, 1997] web site recommender system models web documents using the 128 most informative words. Formally, an item i from a collection I is modeled by the vector $\overrightarrow{Content}(i)$ defined as follows:

$$\overrightarrow{Content}(i) = \langle w_{1,i} \dots w_{n,i} \rangle, \quad (2.5)$$

where $w_{k,i}$ is the k^{th} weight of item i , and indice n is the number of keywords allowed.

One of the best measures to compute the weights of keywords is the *term frequency-inverse document frequency* measure, known as $TF - IDF$. Note that alternative machine learning techniques such as the Minimum Description Length has proven very effective in the past [Lang, 1995], but TD-IDF remains the most popular measure. TF-IDF evaluates how important a keyword is to an item, but also takes into account that frequent keywords that appear in many items are not very relevant. Concretely, TF-IDF works as follows. First, the frequency $f_{k,i}$ of each keyword occurring in item i is counted. Then, the term frequency of each keyword k of item i , $TF_{k,i}$, is computed as such:

$$TF_{k,i} = \frac{f_{k,i}}{\max f_i}, \quad (2.6)$$

where $\max f_i$ is the maximum frequency of all the keywords that appear in item i . The inverse document frequency of the keyword k , $IDF(k)$ is computed in order to see how often k appears in the collection of items i . Assume that the size of collection I is N and that there are n_k items containing keyword k , then $IDF(k)$ can be defined as follows:

$$IDF_k = \log \frac{N}{n_k}. \quad (2.7)$$

Finally, the TF-IDF weight of each keyword is computed by offsetting the term frequency by the inverse document frequency, which is formally calculated as follows:

$$w_{k,i} = TF_{i,j} * IDF_k. \quad (2.8)$$

2. For each user u , a *user profile is learnt*, $\overrightarrow{Profile}(u)$, based on the items u has previously liked. Typically, $Profile(u)$ is computed as an average vector of all the individual content vectors of the items liked by u [Balabanovic and Shoham, 1997]. Formally, the user profile of a user u is computed as follows:

$$\overrightarrow{Profile}(u) = \frac{\sum_{k=1}^{|Liked|} Content(Liked_k)}{|Liked|}, \quad (2.9)$$

where $Liked$ is the set of items that the user previously liked, \sum is defined as the sum of two vectors. Note that statistical methods such as Bayesian classifier¹² can also be used [Pazzani and Billsus, 1997].

3. Finally, *item recommendation* is performed by first computing the similarity between all the items not seen by the user and its user's profile. Thus, the recommendation list is composed of the most similar items to the user's preference profile. Formally, given the vector representation of the items and profiles, the most common technique to compute pairwise similarity is by measuring the cosine angle between them [Balabanovic and Shoham, 1997]. If two vectors are very similar, then the angle between them will be small, which gives a cosine value close to 1 (Figure 2.10). Inversely, the cosine angle of dissimilar vectors is close to 0. Thus, the similarity between an item i and the profile of a user u is defined as:

$$similarity(i, u) = \cos(\overrightarrow{Content}(i), \overrightarrow{Profile}(u)) = \frac{\overrightarrow{Content}(i) \cdot \overrightarrow{Profile}(u)}{\|\overrightarrow{Content}(i)\| \cdot \|\overrightarrow{Profile}(u)\|}. \quad (2.10)$$

Note that the recommender system NewsWeeder uses the standard least-squares regression [Lang, 1995].

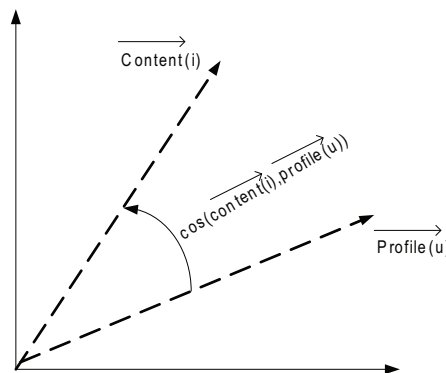


Figure 2.10: Illustration of the cosine angle between an item i and user u in a 2 dimensional space.

¹²Bayesian classifier are explained in details in section 2.3.2

To illustrate this approach, imagine a simple example where a user *Vincent* has liked an article x that can be represented by the keywords: $\langle car, buy, USA \rangle$. Furthermore, suppose that the database contains three items, which are represented by the following keywords:

1. $a = \langle BMW, car, buy, USA \rangle$
2. $b = \langle van, truck, Boston, USA \rangle$
3. $c = \langle buy, truck, USA \rangle$

In order for the system to compare the items, these must be defined by a vector of equal size made of the same keywords. A possible vector could be $v = \langle BMW, car, van, truck, buy, Boston, USA \rangle$. Using vector v , and under the simplifying assumption that all the weights computed using Equation 2.8 are equal to 1, then the user profile and the items can be represented as follows:

- $\overrightarrow{Profile}(Vincent) = \langle 0, 1, 0, 0, 1, 0, 1 \rangle$,
- $\overrightarrow{Content}(a) = \langle 1, 1, 0, 0, 1, 0, 1 \rangle$,
- $\overrightarrow{Content}(b) = \langle 0, 0, 1, 1, 0, 1, 1 \rangle$,
- $\overrightarrow{Content}(c) = \langle 0, 0, 0, 1, 1, 0, 1 \rangle$.

Finally, step 3 computes the similarity between the user profile and all the items to be recommended, and recommend the item with the highest similarity value. Formally, using Equation 2.10, the similarity values between the user profile and items a , b , c , are respectively equal to:

- $similarity(a, Vincent) = \frac{\langle 1, 1, 0, 0, 1, 0, 1 \rangle \cdot \langle 0, 1, 0, 0, 1, 0, 1 \rangle}{\| \langle 1, 1, 0, 0, 1, 0, 1 \rangle \| \cdot \| \langle 0, 1, 0, 0, 1, 0, 1 \rangle \|} = \frac{3}{\sqrt{4}\sqrt{3}} \simeq 0.867$,
- $similarity(b, Vincent) = \frac{\langle 0, 0, 1, 1, 0, 1, 1 \rangle \cdot \langle 0, 1, 0, 0, 1, 0, 1 \rangle}{\| \langle 0, 0, 1, 1, 0, 1, 1 \rangle \| \cdot \| \langle 0, 1, 0, 0, 1, 0, 1 \rangle \|} = \frac{3}{\sqrt{4}\sqrt{3}} \simeq 0.289$,
- $similarity(c, Vincent) = \frac{\langle 0, 0, 0, 1, 1, 0, 1 \rangle \cdot \langle 0, 1, 0, 0, 1, 0, 1 \rangle}{\| \langle 0, 0, 0, 1, 1, 0, 1 \rangle \| \cdot \| \langle 0, 1, 0, 0, 1, 0, 1 \rangle \|} = \frac{3}{\sqrt{3}\sqrt{3}} \simeq 0.667$.

Thus, the information retrieval approach will recommend item a to *Vincent*.

2.3.2 Machine learning approach

Machine learning techniques can also be used to build content-based recommender system. The most common technique is the naive Bayesian classifier [Pazzani and Billsus, 1997, Mooney et al., 1998, Mooney and Roy, 2000], but other more complex techniques such as the basic nearest neighbors algorithm [Duda and Hart, 1973], the nearest neighbor algorithm PEBLS [Cost and Salzberg, 1993], decision tree learners such as ID3 [Quinlan, 1990], the Rocchio algorithm [Ittner et al., 1995], and neural nets [Widrow and Hoff, 1988]. However, [Pazzani and Billsus, 1997] have found that naive bayesian classifiers performed at least as well as the more complex techniques just cited.

As with the information retrieval approach, the naive Bayesian classifier approach is also a three phases process. The first phase is very similar to the information retrieval approach, but the other two are very different. It is the way the user's profile is built that differs. Rather than using a vector of weights, the users' profiles are constructed using the bag-of-words naive Bayesian classifier [Mitchell, 1997].

Note that the Bayesian classifier differs from the information retrieval in two fundamental ways. First, the information retrieval approach sees the recommender problem as retrieving a set of relevant items based on the user's preferences, while the Bayesian approach sees it as a classification problem. Second, the IR approach computes the similarity between an item and the user's profile using the cosine angle, while the bayesian approach makes a prediction on a model learnt on what a user has seen.

The Bayesian classifier sees the recommendation problem as classifying a set of items I into a certain class $c \in C$, where C is set of all possible classes. For example, C can be a set composed of two classes such as $\{relevant, irrelevant\}$, or five classes like $\{1, 2, 3, 4, 5\}$, if the traditional 5 star ratings scale is used. Formally, the naive Bayesian classifier is used to compute the probability that an item i belongs to a certain class c , $P(c|i)$.

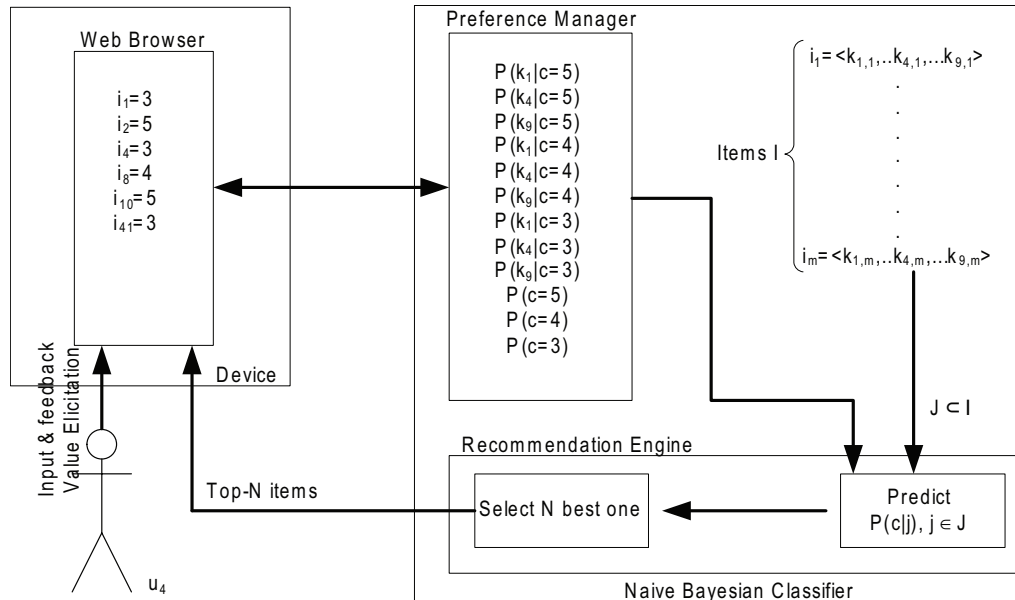


Figure 2.11: The naive Bayesian classifier process.

Figure 2.11 illustrates the Bayesian approach that is composed of the following three phases:

1. The *item representation* is simpler than the one used by the information retrieval technique. Instead of using a vector of the keywords weights, the bayesian approach simply uses the keywords themselves. Thus, an item i is represented as follows:

$$\overrightarrow{Content}(i) = \langle k_{1,i} \dots k_{n,i} \rangle, \quad (2.11)$$

where $k_{k,i}$ is the k^{th} keyword of item i . A major difference with the information retrieval is that the number of keywords is not restrained. In practice however,

systems tends to limit the number of keywords in order to reduce the dimensions [Pazzani and Billsus, 1997], but not always [Mooney and Roy, 2000].

2. In the *learning user profile* phase, the system needs to learn some probabilities based on what the user has previously seen. Formally, using the Bayes Rule, the probability $P(c|i)$ can be computed as follows:

$$P(c|i) = \frac{P(c)}{P(i)} * P(i|c), \quad (2.12)$$

where $P(c)$ is the probability of a given class, $P(i)$ is the probability of that item i occurring, and $P(i|c)$ is the probability that item i appears given a class c . To recommend an item i , the system needs to calculate $P(c|i)$ for each of the class in C , and find the largest probability. As each of these calculations involves the unknown but fixed probability $P(i)$, it can be ignored, which simplifies the model to:

$$P(c|i) = P(c) * P(i|c). \quad (2.13)$$

Under the assumption that each keyword appears independently from each others (which is clearly not true in most situations), the probability $P(i|c)$ can be decomposed as follows:

$$P(i|c) = \prod_{l=1}^{|\overrightarrow{Content}(i)|} P(k_{l,i}|c) = P(k_{1,i}|c) * P(k_{2,i}|c) * \dots * P(k_{n,i}|c), \quad (2.14)$$

where $P(k_{l,i}|c)$ is the probability of having the l^{th} keyword when we have the class c . Given the user preference profile, $P(k_{l,i}|c)$ is calculated as the number of times the l^{th} keyword appears in class c divided by the total number of keywords in c . $P(c)$ is computed as the total number of keywords in class c divided by the total number of keywords in set of classes C .

3. Finally, the *item recommendation* is made by computing all the conditional probabilities $P(c|i)$ for each class c , and item i in I . Note that the largest $P(c|i)$ determines the class of item i . Using the user's probabilistic model computed in phase 2, $P(c|i)$ is computed as follows:

$$P(c|i) = P(c) * \prod_{l=1}^{|\overrightarrow{Content}(i)|} P(k_{l,i}|c) = P(c) * (P(k_{1,i}|c) * \dots * P(k_{n,i}|c)). \quad (2.15)$$

There are two common criticisms concerning the naive Bayesian classifiers. First, it makes the naive assumption that keywords are independent of each other. Second, it requires a lot of training data to estimate the various probabilities. Despite these limitations, and when sufficient training data is available, Bayesian approach has performed reasonably well [Melville et al., 2001].

2.3.3 Preference-based approach

Contrary to machine learning and information retrieval approaches that consider the textual content of an item, a *preference-based approach*, PBA, uses the *attributes* of an item as its content. Moreover, the preference based approach sees the recommender problem as a *multi attribute decision problem*, where the system helps the decision maker to determine the optimal solution from a large set of available outcomes, according to the decision maker's preferences. In the recommender context, the decision maker is the user to whom we want to recommend items, while the outcomes are the set of available items.

Formally, the multi-attribute decision problem in the recommender context is defined by the following tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{I} \rangle$, where

- \mathcal{X} is the set of attributes $\{x_1, \dots, x_p\}$ describing all the items;
- \mathcal{D} is the set of allowed domain values $\{D_1, \dots, D_p\}$, where each D_i is the set of possible values for attribute x_i . Note that the set of possible values for an attribute can be continuous or discrete;
- \mathcal{C} is the set of constraints $\{c_1, \dots, c_p\}$, where each c_i is a constraint function that restrict the values that a subset of \mathcal{X} can take;
- \mathcal{I} is the set of items that we want to recommend to the user, and is within the cartesian product $D = D_1 \times D_2 \times \dots \times D_p$. Note that the set \mathcal{I} is commonly called the available outcomes.

To illustrate this model, imagine that we want to model items for the real-estate domain. Apartments can be modeled using five distinct attributes: $\mathcal{X} = Type, Nbrooms, Kitchen, Bathroom, Size$, and $Price$. The domain value for each attribute may be defined by the following sets:

- $D_{Type} = \{house, apartment\}$;
- $D_{Nbrooms} = [1, 20]$;
- $D_{Kitchen} = \{private, share, none\}$;
- $D_{Bathroom} = \{private, share, none\}$;
- $D_{Size} = [10, 1000] m^2$;
- $D_{Price} = [0, 10000] CHF$;

The domain D_{Type} is discrete and made of two domain values: *house* and *apartment*; while it is continuous for the price attribute, which can take any value in the interval 0 to 10000 CHF. The nature of the real estate domain also implies a set of constraints such as:

- $C_{Type, Kitchen, Bathroom}$:If $Type = house$ then $Kitchen = private \wedge bathroom = private$;
- $C_{Type, Size}$:If $Type = house$ then $size > 70m^2$;

The preference based approach must carefully elicits the user's preferences, and then uses a variant of the *multi-attribute utility theory* to compare the items and find the best ones [Winterfeld and Edwards, 1986]. The root of the utility theory dates from the early 40s when Von Neumann and Morgenstern [Neumann and Morgenstern, 1944] have laid the foundations and proved, under condition of four axioms, that preferences and attitudes towards risk can be adequately modeled by a utility function, u . Formally, let \mathcal{L} be the set of all risky prospects on the set \mathcal{I} , where $\sum p_k i_k \in \mathcal{L}$ and $\sum p_k = 1$. Informally, a risky prospect is an uncertain outcome, which means it has a probability p_k of occurring. Let also \succeq be a binary relation on \mathcal{L} that follows four axioms:

1. \succeq is complete, i.e.: $\forall x, y \in \mathcal{L}: x \succeq y$ or $y \succeq x$;
2. \succeq is transitive, i.e.: $\forall x, y, z \in \mathcal{L}: x \succeq y$ and $y \succeq z \Rightarrow x \succeq z$;
3. Continuity Axiom, if $x, y, z \in \mathcal{L}$ such that $x \succ y \succ z$, then $\exists \alpha, \beta \in (0, 1)$ such that $\alpha x + (1 - \alpha)z \succ y \succ \beta x + (1 - \beta)z$;
4. Independence Axiom, $\forall x, y, z \in \mathcal{L}$ and $\exists \alpha \in [0, 1]$, $x \succeq y \Leftrightarrow \alpha x + (1 - \alpha)z \succeq \alpha y + (1 - \alpha)z$;

If these 4 axioms hold, then the simplified form of the Von Neumann and Morgenstern theorem states that: if prospect i is considered better or equivalent to prospect j , then there exists a utility function u that must satisfy the Equation 2.16.

$$\forall x, y \in \mathcal{L}, x \succeq y \Leftrightarrow u(x) \geq u(y). \quad (2.16)$$

In practice however, the risk associated to an outcome is usually ignored, which simplifies Equation 2.16 as follows.

$$\forall x, y \in \mathcal{I}, x \succeq y \Leftrightarrow v(x) \geq v(y). \quad (2.17)$$

where $v(x)$ is the *value function* associated to outcome x . Note that only when no uncertainty is involved, then the utility function and value function are interchangeable. Moreover, this dissertation does not consider uncertainty, as most recommender systems don't. Thus, from this point onwards and unless stated otherwise, this dissertation will use the term *utility function* to denote a value function.

Furthermore, if we assume that the Mutual Preferential Independence assumptions holds [Keeney and Raiffa, 1993], then the theorem of Additive Value Function can be used, and the utility V of an item i can be defined as the sum of the sub-utility functions v_k of item i on each attribute multiplied by its weight w_k (Equation 2.18). This is commonly called the Weighted Additive Strategy (WADD, [Keeney and Raiffa, 1993, Payne et al., 1988]).

$$V(i) = \sum_{k=1}^p w_k v_k(i) \quad (2.18)$$

Most preference-based recommender systems use the WADD strategy to recommend items to the user. In the recommender context, the sub-utility function v_k reflects how much

a user likes the attribute k , while its weight w_k reflects its importance to the user. It is the set of all sub-utility functions and weights that makes the user's preference profile.

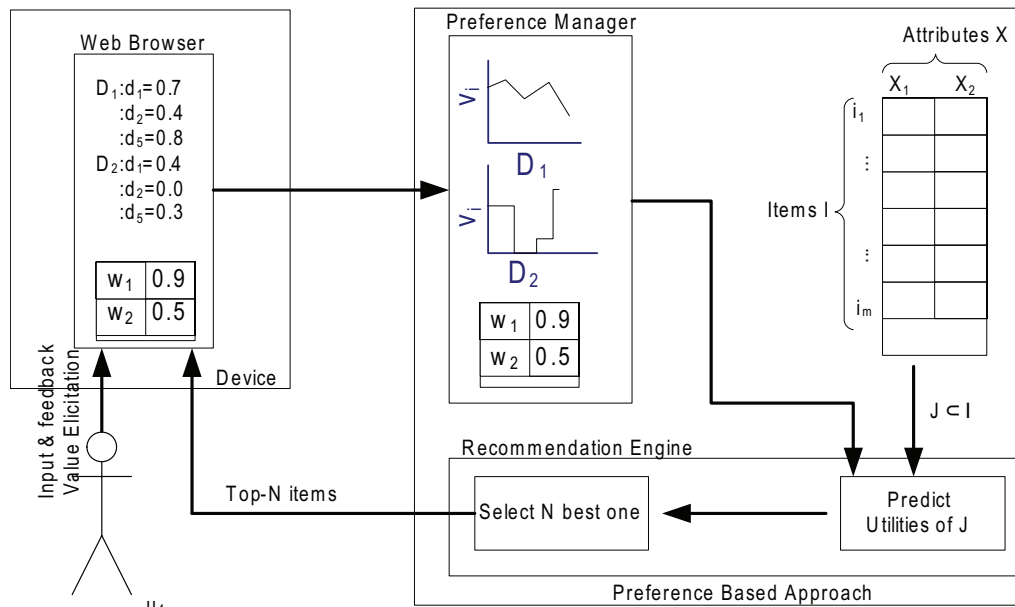


Figure 2.12: The preference-based approach process.

Concretely, Figure 2.12 illustrates the WADD strategy for a recommender system, which can be described as follows:

1. The *preference elicitation* process asks value elicitation questions to the user in order for her to specify some preferred values and the weight associated to each attribute. Imagine an example where a user wants to buy a computer. Figure 2.13 shows a fraction of the web interface that is displayed on the client's web browser. Note how the user expresses her preferences by selecting the *Toshiba* brand for the manufacturer attribute, and a maximum price of \$1500.

The screenshot shows a web interface titled "Please enter your preferences". It has two sections: "Manufacturer" with a dropdown menu set to "Toshiba" and "Importance" with five radio buttons (least, most) where "most" is selected; and "Price" with a dropdown menu set to "<= \$ 1500" and "Importance" with five radio buttons (least, most) where "most" is selected.

Figure 2.13: Illustration of the value elicitation process for the preference-based approach

2. The *utility functions* are then constructed based on the preferences of the user. Constructing these functions is actually a hard problem from an elicitation point of view. The *Midvalue algorithm* which is designed to build such functions requires asking the user a set of mid-value points [Zhang and Pu, 2004]. Unfortunately, this algorithm requires to elicit at least 3 utility values per attribute, which is obviously not feasible in real problems. To overcome this problem, practical applications use a simple two

steps heuristic. First, the system asks for the preferred value for all the attributes (step 1). Then, the system uses predefined basic utility functions and fine-tunes them based on the preferred value given by the user.

Further consider the example where the user set the maximum price of the notebook to \$1500, and set the weight of this attribute to 4 out of a maximum of 5. The system will then choose a predefined function like in Figure 2.14(a), and adapt it to fit the $\leq \$1500$ preferred value. Note that in Figure 2.14(a), the distance between points x and y is usually inversely proportional to the weight of the attribute.

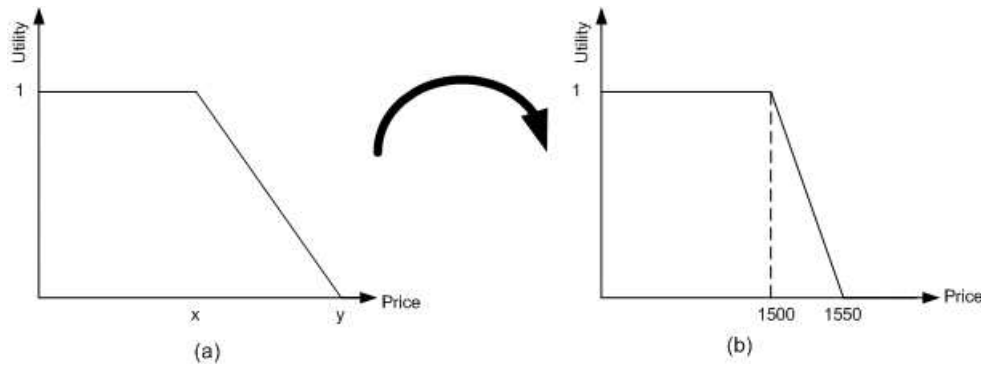


Figure 2.14: Two utility functions associated to the price attribute of Figure 2.13; (a) shows the predefined utility function, and (b) shows the utility function after that a user said that she wants a computer less or equal than 1500\$.

3. The recommender system *evaluates the utility* of all the items in \mathcal{I} using Equation 2.18, where the sub-utility values and weights are taken from the user profile. Finally, the items are sorted by decreasing order of their utility value, and the first N items are displayed to the user.

Theoretically, when the mutual preferential independence assumption holds, and if all the parameters can be precisely elicited, then this strategy can achieve 100% prediction accuracy. This explains why this approach is widely used on the internet; such examples are notebookreview.com, FlatFinder[Viappiani et al., 2006], ActiveDecision (which has been acquired by Knova¹³), and so forth. Unfortunately, the elicitation of the parameters is expensive, and various authors have tried to simplify this elicitation process in order to make it usable in real systems. For example, [Stolze, 2000] exploits the idea of a scoring tree, where a user expresses her preferences by modifying an existing tree via the use of rules. Once the preferences have been elicited, the system translates the scoring tree into a MAUT additive value function, and then searches in the catalog for the most suitable products. Incremental utility elicitation is another approach that eases the elicitation by an incremental process that interleaves utility elicitation and filtering of the items based on the elicited information [Ha and Haddawy, 1997]. A major contribution in that domain is the work done by Ha and Haddawy [Ha and Haddawy, 1997, Ha and Haddawy, 1999], where polyhedral cones and pairwise comparison are used to estimate the user's true weights. Note that, [Ha and Haddawy, 1999] make the assumption that the utility function has a

¹³www.knova.com

multi-linear form, and that all the sub-utility functions are known. Regrettably, computing the cone is a hard problem that makes it unsuitable for real life scenarios. More recently, Blythe [Blythe, 2002] has simplified the process by assuming MAUT additive value functions, and used a linear programming formulation and pairwise comparison of alternatives to estimate the user's true utility. Another approach proposed by [Reilly et al., 2004] is dynamic critiquing to stimulate the user to express more preferences by showing her critique patterns between two examples. Later, [Viappiani et al., 2006] proposed another method for stimulating the user to express more preferences by showing her suggestions that could become optimal under certain circumstances.

Despite its advantages, the preference based approach has three major problems. First, the elicitation process is time consuming and cognitively very complex for the user. Moreover, as argued in section 2.2.2, the value elicitation process used to elicit the user's preferences is not adapted to low-risk domains as found in many eCommerce applications. Second, the recommender system tends to recommend very similar items to the one previously liked by the user. This can be a problem as it reduces the diversity and increases the risk of redundant predications. Take for example a user who said that she liked the 3rd Harry Potter movie. If the recommender system has other Harry Potter movies in its collection, it will see them as very similar to the user's preferences and will recommend them. Unfortunately, if the user has seen the 3rd one, she has probably seen the first two. Third, the rigidity of the model is another serious limitation in real eCommerce applications. By definition, the multi-attribute utility theory requires that all the items are defined by the same set of homogenous attributes X . However, eCatalogs continuously change, as new items with new attributes will frequently appear. However, MAUT is unable to compare heterogenous items with different attributes. For example, imagine an eCommerce shop that sells digital cameras to end-users. Such cameras could be modeled by the attributes $X_{resolution} = \{3MPix, 4MPix, 5MPix, > 5MPix\}$, $X_{zoom} = \{> 3\times, > 5\times, > 10\times\}$, and $X_{imagecapacity} = \{< 50, 50 - 200, > 200\}$. Further suppose that this shop would now like to sell camera phones as both devices capture images. However, the early camera phone uses Integrated CIF Technology, which is not compatible with previous attributes. As a consequence, the shop will have to change its attribute model to be able to sell both devices. This is both time consuming and expensive.

2.4 Collaborative filtering

Collaborative filtering (CF, [Goldberg et al., 1992]), is very different from previous approaches. Unlike content-based recommendation methods that recommend items similar to the one previously bought by the user, collaborative filtering recommend items that *other* similar users liked. Collaborative filtering is based on the subjective evaluations of other users. Furthermore, CF makes the assumption that similar users like similar items. The main motivation behind this approach is that humans do not share computer difficulties in evaluating objects in various *fuzzy* dimensions such as quality, respectfulness, and so forth. Moreover, human can judge novel objects, even when they are very different from what they have encountered before. Unfortunately, in 1992, computer models underlying rec-

ommender systems could not do this as they relied on the content of an item (see section 2.3 for more details).

In collaborative filtering, users state their preferences by rating a set of items, which are then stored in a user-item matrix R . The entire set of ratings is sometimes called *collaborative data*. Formally, this matrix contains all the users' profiles, where the rows represent the users $U = \{u_1, \dots, u_m\}$, the columns correspond to the set of items $I = \{i_1, \dots, i_n\}$, and $R_{u,i}$ is the rating assigned to item i by the user u . It is common practice to denote the average rating of user u by \bar{R}_u , and the average rating of the i^{th} item by \bar{R}_i .

Given the matrix R , two families of collaborative algorithms have been developed [Breese et al., 1998]:

- **Memory-based CF:** uses the entire matrix R to generate a prediction. For example, *user-based CF* predicts the rating of an item by looking at the rated items of the nearest users to the target user.
- **Model-based CF:** does not work directly on the entire matrix R , but starts by developing a model of the user's ratings using various techniques such as Bayesian networks, clustering, or rule-based approach.

2.4.1 Memory-based collaborative filtering

In 1992, the first collaborative filtering technique was introduced in the Tapestry information system [Goldberg et al., 1992]. The idea was to allow humans, known as moderators, to annotate documents, and then use these annotations to retrieve relevant documents. Unfortunately, Tapestry did not allow to aggregate annotations, which meant that a user had to manually choose a specific moderator for selecting the relevant annotations.

Grouplens is the first collaborative filtering system that recommends items by aggregating the ratings of users who share similar interest [Resnick et al., 1994]. This recommender technique is now commonly called the *user-based collaborative filtering*, and is composed of three main parts:

1. The *preference elicitation* process asks the user to state her preferences by rating a set of items. These ratings are then stored in the user-item matrix R .
2. The next critical task is to actually form the neighborhood of similar users, known as the *user's neighborhood*. For each user u , the main goal of the neighborhood formation is to find an ordered list of the k closest users to u . The neighborhood task is in fact divided into two subtasks, which are as follows:
 - (a) The system needs to compute the pairwise *similarities between each pair of users*, $sim(u, v)$, where $u, v \in U$. Two similarity measures have proven very effective in the past:
 - i. The famous *Pearson correlation* that measures the correlation between the ratings of users u and v :

$$sim(u, v) = \frac{\sum_{i \in I} (R_{u,i} - \bar{R}_i)(R_{v,i} - \bar{R}_i)}{\sqrt{\sum_{i \in I} (R_{u,i} - \bar{R}_i)^2} \sqrt{\sum_{i \in I} (R_{v,i} - \bar{R}_i)^2}} \quad (2.19)$$

- ii. The *cosine* function that computes the cosine value between the ratings of users u and v

$$\text{sim}(u, v) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| * \|\vec{v}\|} \quad (2.20)$$

where \cdot is the dot product of vectors defined in cartesian space, $*$ is the dot product between real numbers, and \vec{u} is a vector containing all the ratings of user u . Note that \vec{u} is in fact the row in R that corresponds to user u .

- (b) After computing all the pairwise similarities, the neighborhood of a user u is formed by simply selecting the k users that have the highest similarities with u . This technique is commonly called the *k-nearest neighbors*. Other techniques such as the aggregate neighborhood formation can also be used [Sarwar et al., 2000a]. This technique is very beneficial for very sparse data sets, but does not scale well as it requires the re-computation of a centroid and list of close neighbors at each iteration of the algorithm.
3. The *recommendation* of an item is made by first predicting the rating of each item using the user's previous ratings and u 's neighborhood. Then, the items are sorted by decreasing order of the predicted ratings, and the first N items are shown to the user. Formally, the predicted rating of an item i for user u , $\widetilde{R}_{u,i}$, is computed as follows:

$$\widetilde{R}_{u,i} = \bar{R}_u + \frac{\sum_{v \in N} \text{sim}(u, v) * (R_{v,i} - \bar{R}_v)}{|\sum_{v \in N} \text{sim}(u, v)|} \quad (2.21)$$

where K is the neighborhood of user u , and \bar{R}_u is user u 's average rating.

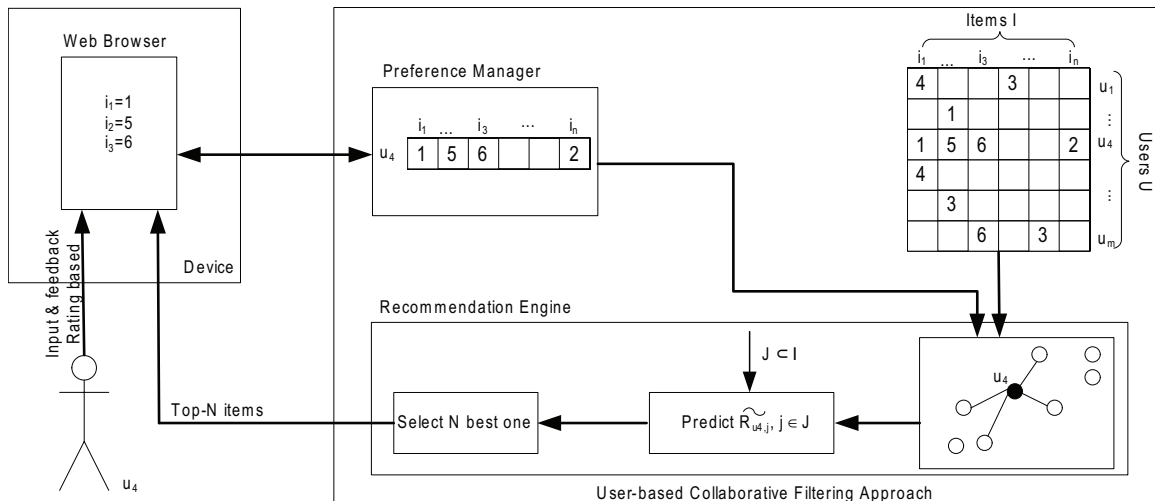


Figure 2.15: The user-based collaborative filtering process.

Experiments on the cinema and newspaper domains showed that the user-based collaborative filtering produces recommendations that are more accurate than content-based approaches that use either traditional information retrieval approach or naive Bayesian classifiers [Melville et al., 2001, Claypool et al., 2001]. Collaborative filtering also allows to recommend novel items to the user, as the recommendation is not made on the item's content, but on whether a similar user as liked the item or not. However, the widespread use of user-based collaborative filtering has been limited due to two major problems:

- **Sparsity:** In practice, users only rate a very small fraction of all available items, maybe 1 to 2%. Thus, the user-item matrix R tends to be very sparse, which limits the neighborhood formation. One approach suggested by [Sarwar et al., 1998] is to use an automated agent, the *filterbot*, that the collaborative filtering system sees as a normal user. Using information retrieval techniques on the item's content, the filterbot evaluates new items as soon as they are published, and stores the results in the user-item matrix R . Other dimensional reduction techniques have also been proposed to reduce the number of entry in the user-item matrix. The most commonly used technique is the Singular Value Decomposition that produce a low dimensional representation of R [Sarwar et al., 2000b]. These techniques have showed significant improvement in the prediction accuracy, but it requires training for selecting the size of the new dimension.
- **Scalability:** The nearest neighbor calculation grows with both the number of users and the number of items. In eCommerce environment where systems have millions of users, the nearest neighbor calculation does not scale well. [Sarwar et al., 2002] proposed to reduce the dimension of search by first clustering the data contained in the user-item matrix, and then forming the neighborhoods from the partitions. Experimental results showed reduction in the complexity, but at a cost in degradation in the recommendation accuracy.

2.4.2 Model-based collaborative filtering

Probabilistic approach

From a probabilistic perspective, the collaborative filtering task can be viewed as calculating the expected rating of an item, given the user's preferences. Formally, the expected rating of item i for user u , $E(R_{u,i})$, is defined as follows:

$$E(R_{u,i}) = \sum_{k=1}^m Pr(R_{u,i} = k | R_{u,l}, l \in LS)k, \quad (2.22)$$

where LS is the set of items that the user u has previously rated, k is the value of a vote, and the rating scale is defined on the interval $[0, m]$. [Breese et al., 1998] proposed two distinct approaches to compute these joint probabilities:

- The *cluster model* exploits the idea that there are certain groups of users that capture a common set of tastes. These groups of users will form the various clusters. The cluster model then partitions each user into a set of clusters. Under the assumption that

the preferences are independent in a given class, [Breese et al., 1998] proposed to use the famous Expectation Maximization algorithm, (EM, [Dempster et al., 1977]), to learn the various clusters using the data found in the user-item matrix R . However, [Ungar and Foster, 1998] argued that the EM algorithm was not suitable for the movie domain, as it didn't recognize the constraints that a movie liked by two different persons must be in the same movie class each time. Thus, the authors proposed to use classical clustering algorithms like K-means and Gibbs instead of EM. Other clustering approaches consist in clustering the users into cliques by recursively calling the K-Means [Castagnos and Boyer, 2006], or on both the users and the items [Kohrs and Merialdo,].

- The *Bayesian network* models each item of the domain as a node in a graph, where the state of each node corresponds to the possible value for each item. Then, a learning algorithm is used to build the Bayesian network. This algorithm searches over various model structures in term of dependencies between each item. In the resulting network, each item has a set of parent items that represent the best predictors for the expected rating.

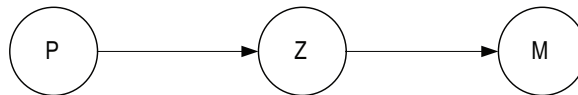


Figure 2.16: Graphical aspect model of the movie domain, where P , Z , and M are random variables that respectively model the objects p , z , and m .

Another probabilistic approach is the *Aspect model* [Hofmann, 1999], which consists in finding a hidden variable between pairs of observations. Take for example the movie domain, where a person p watches a movie m . The observation can be modeled by the tuple (p, m) . The aspect model supposes the existence of a hidden cause z that motivates p to watch m (Figure 2.16). In our movie domain example, a hidden cause z could be the genre of the movie. According to the semantic of the aspect model, a person p chooses the variable z , which in turn determines the movie m to watch. Furthermore, the choice of the movie m is assumed independent of p given the knowledge of z . Thus, a joint probability model of the user and movie can be parameterized by:

$$P(p, m) = \sum_z P(p)P(z|p)P(m|z), \quad (2.23)$$

where parameters $P(z|p)$ and $P(m|z)$ correspond to the processes of p stochastically choosing the latent variable z , and z stochastically choosing m . As for the cluster models, these parameters can be estimated using the EM algorithm. [Schein et al., 2002] also proposed to use the aspect model on the content of the item (the actors) rather than the item itself. Thus, the observations are changed from tuple (p, m) to (p, a) , where a represents an actor. Note that these newly formed observations are no longer independent, which breaks an assumption of the aspect model. However, experiments have shown it to perform well, and even overcoming traditional naive Bayesian classifiers.

Neural networks have also been applied with success [Billsus and Pazzani, 1998], and showed significant improvement over traditional user-based recommender algorithms. Unfortunately, it requires building and maintaining a neural network for each user, which do not scaled when considering real recommender systems. Other statistical techniques include the probabilistic relational model [Getoor and Sahami,], the maximum entropy model [Pavlov and Pennock, 2002], and two-layer undirected graphical models such as restricted Boltzmann machines [Salakhutdinov et al., 2007].

It is often argued that the statistical models are inappropriate for recommender systems, as their independence hypothesis rarely holds, and they usually require a lot of training data in order to estimates the various probabilities. For example, [Breese et al., 1998] has shown that simple correlation based techniques performed better when only limited data was available. Moreover, both [Melville et al., 2001] and [Schein et al., 2002] used over 40 ratings per user to train the system. In practice however, it is unfeasible to assume that a new user is willing or capable to state that many ratings. This is commonly known as the *cold-start problem*.

Machine learning

Machine learning techniques try to find low rank approximation of the user-item matrix R in order to reduce the noise induced by the high sparsity of the data. The most common approach to find such approximation is by using a matrix factorization approach.

Matrix factorization, MF, is a very simple techniques that decomposes the big matrix R into two smaller matrices. Formally, given the user item matrix R of size $U \times I$, MF tries to find two matrices S and T such that $R \approx ST^T$, where U and I respectively represent the users and the items, $S \in \mathbb{R}^{U \times l}$, $T \in \mathbb{R}^{I \times l}$, and $l \ll U, I$. The matrices S and T can be seen as compact representation of the users and the movies, while ST^T contains the ratings assigned be the users in S on items in T .

Singular value decomposition, SVD, is the most popular matrix decomposition technique. SVD states that there exists a factorization of the matrix R such that $R = S\Sigma T$, where $S \in \mathbb{R}^{U \times \min(U,I)}$, $T \in \mathbb{R}^{I \times \min(U,I)}$ are orthogonal matrices, and $\Sigma \in \mathbb{R}^{\min(U,I) \times \min(U,I)}$ contains the singular values on the diagonal. By only keeping the largest k singular values and setting the others to 0, a rank- k approximation of R can be obtained. Note that more advanced matrix factorization techniques such as Bayesian approach [Lim and Teh, 2007] and simple gradient descent [Takacs et al., 2007] also exist.

Unfortunately, two problems arises with the SVD technique. First, standard algorithm for computing the SVD decomposition have a time complexity equals to $\mathcal{O}(\max(U^3, I^3))$. Second, it assumes a fully observed matrix R . These two problems make this technique inappropriate in the recommender systems context, where R contains millions of entries and where missing values cannot be estimated by 0 values. To overcome the complexity problem, [Brand, 2003] proposed an $\mathcal{O}(U \times I \times n)$ algorithm that can approximate the rank- n SVD through rank-1 update. For the second problem, researchers have proposed many solutions ranging from greedy residual fitting [Lim and Teh, 2007] to the classical expectation maximization optimization [Srebro and Jaakkola, 2003].

Item-to-item correlation

Due to the amount of data required to train the statistical models and the strong hypothesis of the underlying data structure, the statistical approaches introduced in Section 2.4.2 have rarely been used in practice. Machine learning techniques also requires a lot of training data in order to estimate the various parameters and can easily suffer from overfitting. However, there is one model based approach that has become very popular, and is widely used on the web - the *item-based collaborative filtering*. For example, Amazon.com¹⁴, with its 29 million customers and several million catalog items, uses item-based collaborative filtering to recommend items to the user [Linden et al., 2003]. Figure 2.17 shows the recommendations made by Amazon.com, which is commonly known to end-users as "Customers who bought this item also bought:", which is illustrated by



Figure 2.17: Example of the item-based collaborative filtering for a user who is looking for the movie Apollo 13

The fundamental difference between the user-based collaborative filtering and the item-based one lies in the objects considered. The former approach works on groups of similar users, while that latter considers groups of similar items. Concretely, the first step of item-based collaborative filtering is the construction of the item-to-item similarity matrix S . This is achieved by computing the pairwise similarity between each pair of items i and j in the matrix R , $sim(i, j)$, using the adjusted cosine metric. The adjusted cosine metric is an improved version of the cosine based approach, which takes into account the difference in rating of each user's profile. Formally, the adjusted cosine metric is defined as follows:

$$sim(i, j) = \frac{\sum_{u \in U} (R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U} (R_{u,i} - \bar{R}_u)^2} \sqrt{\sum_{u \in U} (R_{u,j} - \bar{R}_u)^2}}. \quad (2.24)$$

The Pearson correlation and cosine metric defined in section 2.4.1 can also be used to compute these pairwise similarities. However, it is necessary to consider the columns of matrix R as rating vectors rather than the rows.

Once S has been constructed, the predicted rating of an item i is computed by selecting the k most similar items in S , and then using a weighted average based on the similarity between the selected items and i . Formally, the prediction of an item is computed as follows

¹⁴www.amazon.com

[McLaughlin and Herlocker, 2004, Park et al., 2006]:

$$\tilde{R}_{u,i} = \bar{R}_i + \frac{\sum_{j \in K} (S_{i,j} * (R_{u,j} - \bar{R}_j))}{\sum_{j \in K} |S_{i,j}|}, \quad (2.25)$$

where K is the set of the k closest items to item i , and $S_{i,j}$ is the similarity between items i and j . Note that the item prediction was initially computed using a simpler weighted average combination that did not take into account the item's average rating [Sarwar et al., 2001]. Unfortunately, this simple combination is not very robust to the size of the item's neighborhood, which explains why Equation 2.25 is now used.

Once all the possible items to be recommended have been rated, item-based CF simply selects the N items with the best predicted ratings. This selection strategy is commonly known as the *top-N recommendation strategy*. The item-based collaborative filtering is illustrated in Figure 2.18.

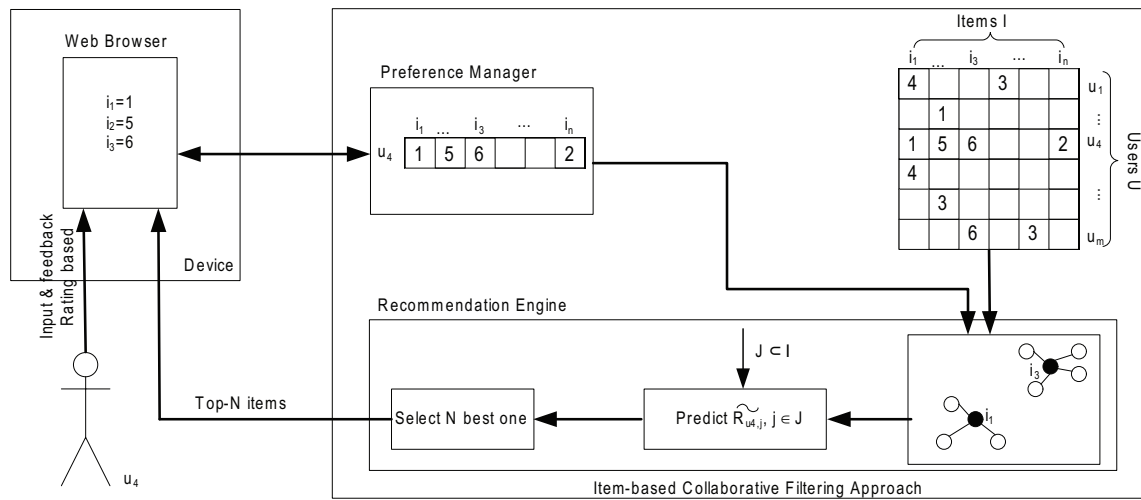


Figure 2.18: The item-based collaborative filtering process.

By working on groups of items rather than groups of users, the item-based collaborative filtering has two main advantages over classical user-based approaches. First, as the number of items is usually much smaller than the number of users in real applications, it makes this approach more scalable. Note also that the items changes less frequently than the users' preference profiles. Second, experimental results have shown it to be more accurate than traditional user-based approaches [Sarwar et al., 2001].

In practice, CF is the most popular recommendation technique and this is due to three main reasons. First, when sufficient preferences from the user are available, studies have shown it to have reasonably good prediction accuracy. Second, the cognitive requirement on the user is very low. Finally, it can recommend items without modeling them, as long as they have been previously rated. However, it has been argued by many authors [Li et al., 2005, Mobasher et al., 2004, O'Sullivan et al., 2004] that Collaborative filtering suffers from the following profound problems:

Sparsity - This problem occurs when the number of items far exceeds what an individual can rate. Thus, the user-matrix R tends to be very empty, which limits the neighborhood formation. Take for example Amazon.com, it has several millions of customers

and users, which gives a database of approximately 10^{12} entries. However, a typical customer only rates about a dozen of recommendations, which gives a density smaller than 0.001%. This is one of collaborative filtering's fundamental problem and explains why numerous authors [Melville et al., 2001] [Mobasher et al., 2004] [O'Sullivan et al., 2004] have focused their work to try to overcome it using *hybrid recommender systems*. Data-mining [O'Sullivan et al., 2004] or the Two-way Aspect Model [Schein et al., 2002] are now used to extract the item similarity knowledge by using association between the user's profile [O'Sullivan et al., 2004] and the object's content [Schein et al., 2002] in order to augment a standard similarity matrix. Singular value decomposition can also be used to reduce the dimension of the user-item matrix R [Sarwar et al., 2000b]. More recently, [Middleton et al., 2004] and [Ziegler et al., 2004] proposed to use a taxonomy for propagating known user's preferences in order to fill in some empty values.

Scalability - The computation of the neighborhood requires looking at all the items and users in the systems. Consequently, as the number of users grows, so does the complexity. By splitting the user-item matrix R , clustering algorithms have proven very effective for reducing the scalability, but can lead to a decrease in decision accuracy [Connor and Herlocker, 2001, Castagnos and Boyer, 2006, Kohrs and Merialdo,].

Cold start - To build a model, the system must know enough preferences about the user. However, a new user is usually not willing to make the effort of rating a sufficient number of items.

Latency or the new item problem - Product catalogs evolve over time; however, the collaborative approach cannot deal with new products that have not been previously rated. To overcome the latency problem, the content of the object has also been used to try to predict its rating. This is achieved by filling up the similarity matrix S with estimated rating based on the content of the item [Claypool et al., 1999], or simply by using a weighted combination of the content and collaborative prediction [Melville et al., 2001] .

Privacy - In most systems, all the users' ratings are located on a server and are accessible to third parties, thus raising serious privacy concerns. To reduce this problem, [Polat and Du, 2003] propose to add random noise to the user's preference ratings, while [Canny, 2002] distributes the collaborative filtering process over the users.

Shilling attacks or profile injection attacks - Malicious users can influence the recommendations by inserting untruthful ratings. Two techniques for corrupting the user-item matrix R have been identified: *push-attack* and *nuke-attack* [O'Mahony et al., 2004]. The first technique consists in inserting profiles in R in order to favor a given item, while inversely, the second is designed to make an item less recommendable. These two techniques rely on the *random* attack and on the *average* attack. The random attack uses the overall distribution of ratings in R , while the average attack uses the average for all the users. To identify these attacks, [Burke et al., 2006] introduced a model-based approach for identifying suspicious user profiles, while [Bhaumik et al., 2006] focused on identifying items that are under attack.

Note that the first two problems also occurs in the traditional user-based collaborative filtering. In practice however, item-based collaborative is usually more scalable than the user-based approach as most recommender systems have less items than users. Moreover, items tend to change less frequently than user's preferences, which reduces the number of time the neighborhood as to be regenerated.

2.5 Knowledge-based recommender systems

Knowledge-based recommender systems use the knowledge about both the users and the items. Surprisingly, this approach has been hardly looked at, and has been overshadowed by the (relative) success collaborative filtering.

Moreover, one of the only knowledge-based recommender system ever developed is the conversational recommender system *FindMe*. Following this, most researchers consider that knowledge-based recommender systems are in fact conversational recommender systems. However, this dissertation does not make this simplification as we show that ontology filtering is a knowledge-based recommender system that does share the properties of the conversational approach.

2.5.1 Conversational recommender systems

One of the first knowledge-based recommender systems is the *FindMe* system, which has been made famous in the Entree restaurant recommender [Burke et al., 1997, Burke, 2000].

The main idea is to use a form of feedback, known as *critique*, in order to stimulate the user to state more preferences. Instead of asking the user specific values, the system presents a set of critiques that are associated with the recommended items, which the user can select in order to refine her search. A typical recommendation cycle in the Entree system is as follows. First, the user either selects a known restaurant from a list, or states some basic preferences such as the type of cuisine she wants, the price, the atmosphere and so forth. If preferences were entered, then the system recommends a restaurant that is the most similar to the user's preferences. Instead, imagine the situation where a user chooses the restaurant "Chinois on Main" in Los Angeles. As shown in Figure 2.19, the system finds the user's restaurant, but also shows a similar one to stimulate her to state more preferences. At the bottom of the page, the user has the option to state more preferences by critiquing the recommendation via the use of one of the seven buttons. Each button corresponds to a critique, where *less \$\$* allows the user to look for a cheaper restaurant. This process iterates until the user finds an appropriate restaurant.

The critiques help the user navigate through complex collections of items, without having to know the underlying features making them. This family of systems is also commonly called *conversational recommender systems*, as it duplicates the conversation a user may have with a seller. Conversational recommender systems make use of two strategies for eliciting the user's preferences [Simazu et al., 2001]:

- *navigation by asking* - the user is asked to give explicit preferences through value elicitation questions (see Section 2.2.2). In the Entree system, this corresponds to the first step, where a user is asked what kind of food or the price range of her budget.

Entree Results

The Los Angeles restaurant you chose is:

Chinois On Main
2709 Main St. (bet. Rose Ave. & Ocean Park Blvd.), Santa Monica, 310-392-9025

| | |
|------------------|-----------|
| Pacific New Wave | \$30-\$50 |
|------------------|-----------|

Extraordinary Decor, Extraordinary Service, Near-perfect Food, Hip Place To Be, On the Beach, Great for People Watching, Parties and Occasions, Weekend Brunch, Weekend Lunch, Fabulous Wine Lists

We recommend:

Yoshi's Cafe
3257 N. Halsted St. (Belmont Ave.), Chicago, 312-248-6160

| | |
|-------------------------------|-----------|
| Asian, Japanese, French (New) | \$30-\$50 |
|-------------------------------|-----------|

Extraordinary Decor, Extraordinary Service, Near-perfect Food, Need To Dress, Prix Fixe Menus, Quiet for Conversation, Very Busy - Reservations a Must, Romantic, Good Out of Town Business, Fabulous Wine Lists, Game, Parking/Valet

less \$\$ nicer cuisine
Less \$\$
traditional creative livelier quieter

Figure 2.19: Illustration of the Entree recommender system.

- *navigation by proposing* - the user is asked to give preferences implicitly via the use of predefined options. Critiques used in Entree is a typical example of a predefined option that allows the user to state preferences implicitly. In Figure 2.19, the recommended restaurant has a price range of 30 to 50 dollars. By using the *less \$\$* critique, the system will translate this implicit price preference into $< 50\$$ explicit preference.

In the first version of the FindMe system, the critiques were designed over a single feature of the items. In [Burke, 2000], compound critiques were introduced that allow the user to critique an item over multiple features. These critiques allow users with limited knowledge of the product domain to easily find the right item that matches their needs. However, the fact that all the critiques must be predefined by hand is a serious limitation to the FindMe system. Moreover, the critiques that are presented are always the same, and are independent of both the user and of the state of the recommendation cycle. To overcome these two problems, [Reilly et al., 2004] introduced *dynamic critiquing*, which allows the compound critiques to be generated and selected during each recommendation cycle. Furthermore, experimental results showed significant reduction of the recommendation cycle. The idea is to use the famous Apriori algorithm to generate the compound critiques from simple unique critique. In short, the Apriori measures the importance of a rule (i.e: critique) in terms of its support and confidence. Suppose the rule $A \rightarrow B$, which means that from the presence of a certain set of critiques A , one can infer the presence of another set of critiques B . Given this rule, rule support is defined as the number of items that contain both A and B divided by the total number of items, while confidence is the percentage of

items containing B given that A occurs. More details about the Apriori algorithms can be found in [Agrawal et al., 1996].

Knowledge based recommenders usually rely on case-based reasoning systems, CBR, to find similar items to the user's preferences. In traditional CBR, the system solves a problem by reusing the solution of previously solved problem it has in its database. Thus, the success of the CBR technique relies on the identification of the right case for a given problem using a similarity metric. For example, the Entree recommender employs such technique to find the best restaurant that is the most similar to the user's preference by using feature to feature matching. More complex similarity metrics have been proposed such as the the Apriori algorithm [O'Sullivan et al., 2004], or a combination of feature by feature matching and concept proximity in an ontology [Bradley et al., 2000], or even by adding diversity to the retrieved cases [Smyth and McClave, 2001].

2.6 Usage of taxonomies in recommender systems

This dissertation proposes a new recommender system that uses an ontology in order to structure the collection of items in the catalog, and to infer missing preferences of a user. The use of taxonomies in recommender system is not novel as such, but it has never been used as a recommender technique by itself.

The idea of using taxonomies for retrieving information is not new either. Traditional information retrieval systems rely on a vector of keywords to identify and retrieve documents (see Section 2.3.1). As a user states her preferences by giving a (small) set of keywords, it usually happens that these keywords are not present in any of the documents. Thus, no document can be retrieved. Inversely, if the system does find some documents, the keywords approach makes it hard for the user to refine her search query if she is not satisfied with the returned documents.

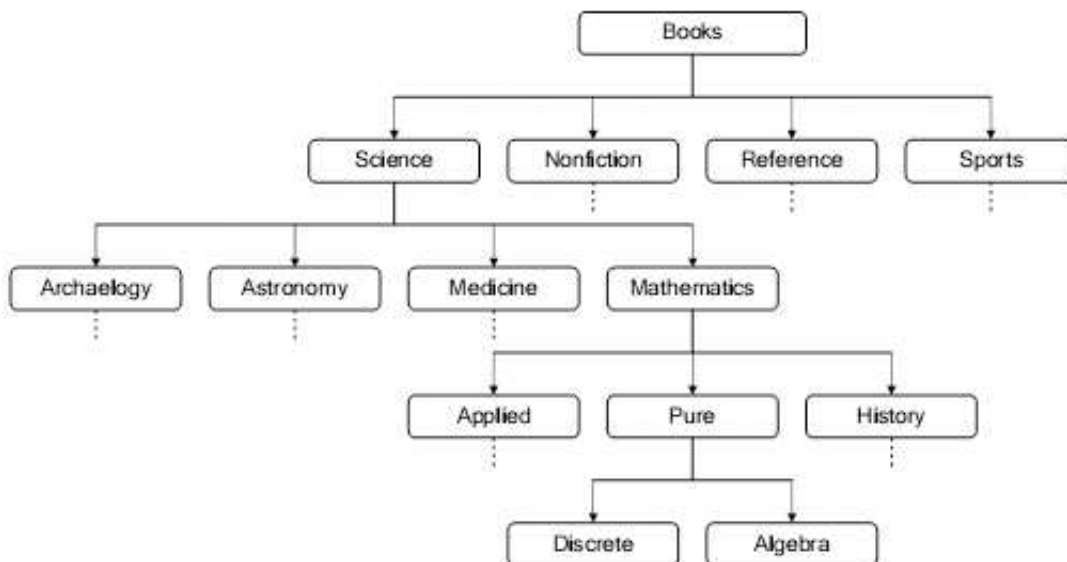


Figure 2.20: Fragment from the Amazon.com taxonomy.

Many authors have realized that the simple keyword approach was not sufficient, and some proposed the use of a *taxonomy* to improve it. A taxonomy is the simplest form of ontology, where a node models a concept and an edge models an inheritance relationship between two concepts. Figure 2.20 illustrates a fraction of the Amazon.com taxonomy, where a rectangular box models a concepts, and an arrow represents an inheritance edge between two concepts. Take for example the *Books* concept, this concept is known as the root of the taxonomy as it has no parent concept. On the other hand, the concept *Discrete* is known as a leaf as it has no child. The concept *Pure* is the parent of the *Discrete* concept.

Given this kind of taxonomy, authors have proposed to enrich the classical information retrieval methods. For example, [Zhuang, 2001] helps the user retrieve picture by using the taxonomy to compute the similarity between keywords that represent the user's query and the pictures. This allows to find pictures that are closely related to the user's query, even if they do not share any keywords. Imagine the case where a picture is annotated with the keyword *discrete*. Further imagine that the user is looking for a picture modeling algebra, which is represented by the query *algebra*. If traditional keyword methods such as TF-IDF was used, this would not return any picture as the keywords do not overlap. However, these two keywords are very similar (given the taxonomy in Figure 2.20), which allows the system to retrieve some pictures. [Liu and Lieberman, 2002] use the concept expansion approach to expand a photo's annotations in order to augment the number of keywords made available to the information retrieval systems. Given a starting concept, a spreading activation algorithm is applied that activates some of the neighborhood concepts, and then reiterates until no more concept is activated [Anderson, 1983]. These activated concepts are then used to annotate the photo. Taxonomies are fundamental elements of the semantic fisheye view technique [Janecek et al., 2005], which help a user perform opportunistic search in an annotated image collection. In traditional case-based reasoning, taxonomies have also been used to help compute the similarity between cases [Bradley et al., 2000]. Taxonomies with more complex relations are also at the root of the Semantic Web¹⁵, which allows the user to use the semantic contained in the ontology to perform smarter searches [Guha et al., 2003, Davies and Weeks, 2004, Sieg et al., 2005].

The novelty of this dissertation is to directly use the ontology not only to build a complete user's preferences profile, but also to make the recommendations. Note that previous authors simply used the taxonomy to fill in some missing values in the user's preference profile, and then used traditional collaborative filtering to make the recommendations. For example, [Middleton et al., 2004] introduced the idea of representing user's preference profiles in ontological terms for recommending research papers. As with the Information Retrieval approach, the research papers are represented by a vector of terms. These terms represent the research topic that index the documents, and simple term frequency is then used to calculate the terms weights. In parallel, they use a taxonomy of research topics to classify each document in a given concept. The user interest profiles are generated based on the research paper a user has browsed. From each browsed paper, the research topics are extracted and used to build a vector of research topics. Their novel idea was then to use the ontology to propagate 50% of the value of a topic of interest to its super-concept in order to fill up the user interest profile vector. The recommendations are then made using

¹⁵<http://www.w3.org/2001/sw/>

a collaborative filtering approach, where the recommendations are formulated between the users' current topics of interest and the papers classified in these various topics.

Figure 2.21 illustrates the approach used by Middleton et al. to overcome the sparsity. As one can see, the value 6 assigned to a concept is propagated to its parent as it does not contain any value. This mechanism allows to reduce sparsity in the user's profile, which increases the accuracy of collaborative filtering. However, the propagation is always 50% of the initial value, which assumes that each concept is uniformly distributed in the ontology. Obviously, this hypothesis does not hold in real life.

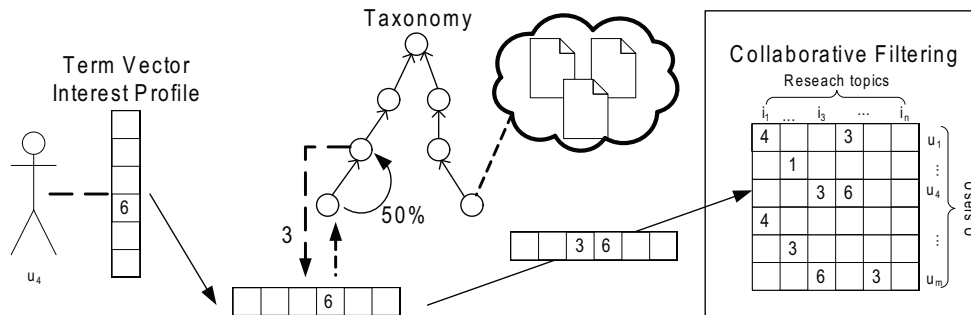


Figure 2.21: Approach used by Middleton et al. for reducing sparsity in collaborative filtering.

[Ziegler et al., 2004] extended the work of Middleton et al. by increasing the range of the propagation. Instead of only propagating the interest value from a concept c to its parent, Ziegler et al. proposed to propagate it among all the concepts found on the path between the concept c and the root of the taxonomy. Obviously, this mechanism allows to further reduce the sparsity of the term vector but at a cost in scalability. Note that this idea is very similar to the spreading activation algorithm, with two exceptions. First, the propagation is only done upwards. Second, the propagation is not stopped by an activation threshold, but instead finishes when it reaches the root of the tree. Unfortunately, Ziegler et al. make the assumptions that sub-concepts have equal shares in their super-concept, and that there exists a propagation factor k that guides the amount of propagation from a concept to its parent. Even if the first hypothesis does not have a big impact in practice, the second clearly limits the usage of their approach in real life as this coefficient k needs to be learnt. As with Middleton et al. approach, collaborative filtering is then used to generate the recommendations. Finally note that both of these approaches rely on the fact that the topology of the taxonomy is a tree, and that it already exists. This work does not make such assumptions, and proposes algorithms that can automatically learn these taxonomies from past users' experience. Moreover, the next chapter shows that it is possible to extract useful information from these taxonomies in order to transform them into meaningful ontologies.

2.7 Collaborative filtering vs. Preference-based approach

Today, collaborative filtering and the preference-based approaches are the most widely used techniques by eCommerce vendors. For example, collaborative filtering is being success-

fully used by Amazon.com and WalMart.com, while notebookreview.com and ActiveDecision use the preference-based approach. This chapter has introduced these two techniques in details, and this section will now summarize their advantages and drawbacks in order to motivate the need of a new one. Table 2.1 summarizes the tradeoffs between these two recommendation techniques.

| | Input Data | Advantages | Disadvantages |
|-----|--|--|---|
| CF | Ratings on some items | No domain knowledge needed Low cognitive requirement Accuracy improves over time Implicit feedbacks sufficient Good domain discovery | Big training data Cold-start problem Sparsity problem Latency problem Scalability problem Shilling attacks Privacy concerns |
| PBA | Utility functions and weights on well defined attributes | Good Accuracy No training data | Preference elicitation Static model Domain knowledge Highly cognitive |

Table 2.1: Brief tradeoffs analysis between collaborative filtering and the preference based approach.

Two major differences appear between these two approaches. First, collaborative filtering needs a significant amount of training data to build a model of the users (i.e.: the item-to-item similarity matrix and the items' neighborhoods). As the preference based approach works only with the user's own preferences, no such training data is required. Second, the preference based approach requires a user to express preferences directly on the attributes of the items. Thus, this requires good domain knowledge from the user and high cognitive reasoning. As collaborative filtering reasons over ratings, this problem is avoided.

If the user's preferences can be fully elicited, then preference based approach can theoretically reach 100% prediction accuracy. However, eliciting utility functions and weights is a very tedious process. This explains why the preference based approach is only used in medium to high user involvement decision processes, where the risk of failure is high. For example, when someone wants to buy a house, this person is usually ready to spend time expressing her preferences, as the cost of failure is much higher than buying a \$10 CD. On the other hand, the risk in choosing a movie is much lower and does not justify a large effort. Moreover, these preferences must be defined over a set of well defined attributes. In many eCommerce environments, eCatalogs are heterogenous and attributes are usually not well defined, which limits the use of the preference based approach. Collaborative filtering does not use the items' attributes, instead it ranks items based on the ratings assigned by other users. This flexibility allows CF to be used in heterogenous environments, while requiring only low cognitive reasoning from the user. This advantage has made collaborative filtering very popular on the web. For example, Amazon.com and YouTube use collaborative filtering to respectively recommend books and videos to their users.

Unfortunately, collaborative filtering needs to elicit many ratings from the user in order to achieve reasonable prediction accuracy. We believe that this cold start problem is due

to the unstructured item space, as items are seen as flat objects. To illustrate this problem, reconsider the little DVD renting problem introduced in the introduction section. Figure 2.22(a) shows the user-item matrix R that contains the preferences of three users: David, Paolo, and Alex. The ratings range from 1 to 5, where 1 means that the user strongly disliked the item, and inversely 5 means that she loved it. The symbol x denotes that no preferences were stated by the user. Let's now imagine that Alex is planning to rent a DVD tonight.

| | DVD ₁ | DVD ₂ | DVD ₃ | DVD ₄ |
|-------|------------------|------------------|------------------|------------------|
| David | 4 | 5 | x | x |
| Paolo | x | x | 5 | 4 |
| Alex | 4 | x | x | 4 |

(a) user-item matrix R

| | DVD ₁ | DVD ₂ | DVD ₃ | DVD ₄ |
|------------------|------------------|------------------|------------------|------------------|
| DVD ₁ | 1 | -1 | x | 0 |
| DVD ₂ | -1 | 1 | x | x |
| DVD ₃ | x | x | 1 | -1 |
| DVD ₄ | 0 | x | -1 | 1 |

(b) item-item matrix S

Figure 2.22: (a) matrix R , while (b) is matrix S computed from (a).

To use the item-based collaborative filtering, the system starts by constructing the item-item similarity matrix S , where S is constructed from R using Equation 2.24 (Figure 2.22(b)). Given Alex's preferences, the predicted rating of an item i is computed using a weighted average of Alex's previous ratings by the similarity of closest neighbors to i . Formally, the predicted rating of item i , $\hat{R}_{Alex,i}$, is computed as follows:

$$\hat{R}_{Alex,i} = \bar{R}_i + \frac{\sum_{j \in K} (S_{i,j} * (R_{Alex,j} - \bar{R}_j))}{\sum_{j \in K} |S_{i,j}|}, \quad (2.26)$$

where K is the set of the k -closest neighbors to item i . If we set $|K|$ to 2, the predicted ratings of items DVD_2 and DVD_3 become 4 (i.e.: $\hat{R}_{Alex,DVD_2} = \hat{R}_{Alex,DVD_3} = \bar{R}_{DVD_3} + (S_{DVD_3,DVD_4}(R_{Alex,DVD_1} - \bar{R}_{DVD_4}) + S_{DVD_3,DVD_1}(R_{Alex,DVD_1} - \bar{R}_{DVD_1})) / (S_{DVD_3,DVD_4} + S_{DVD_3,DVD_1}) = 5 + 0 = 5$). As both items have identical predicted ratings, CF would recommend both to Alex. To discriminate these two DVDs, CF would have to elicit more ratings from Alex, which Alex might not be willing or able to give.

Moreover, the k -nearest neighbors algorithm used by most collaborative filtering techniques have three major drawbacks. First, the parameter k that determines the size of the neighborhood has to be learnt, and the optimal value constantly varies depending on the user's preferences and others. Moreover, [Herlocker et al., 1999] and experiments in Section 8.3.5 have shown that if too many low correlated neighbors are picked, then the prediction accuracy actually decreases. Last but not the least, this algorithm is not very scalable as the complexity increases with the number of neighbors k .

2.8 Ontology filtering

To summarize, the lack of appropriate model for representing the eCatalog, and the preference elicitation overload are identified as the two main reasons why traditional recommender systems failed to meet users' expectations. This dissertation proposes to use an

ontology to add a structure to the collection of items, and to use this ontology to infer missing user's preferences. The ontology allows to reduce the search space, and also ease the preference elicitation process, as missing information will be inferred.

This dissertation presents a novel knowledge based recommender system called *ontology filtering* that can overcome the mentioned problems. Ontology filtering differs from existing techniques for the following reasons:

- Ontology filtering does not use collaborative filtering nor a content based approach to make the recommendations, but uses the knowledge of the items contained in an ontology and the user's preferences. Thus, ontology filtering is in fact a knowledge based recommender system.
- Ontology filtering proposes an algorithm that is capable of learning a set of ontologies if none are available. Furthermore, another algorithm is proposed that is capable of personalizing the ontology based on the user's preferences.
- Given the value of a single concept c , ontology filtering is capable of inferring the value of any other concepts in the ontology. Note that the approach proposed by [Ziegler et al., 2004] can only infer the value of concepts that are parent of c .
- The inference mechanism proposed by ontology filtering does not require any parameter tuning, but only uses the user's preferences and the topology of the ontology. [Ziegler et al., 2004] requires a propagation factor k in its inference process that needs to be learnt.

The components making ontology filtering are introduced as follows. First, chapter 3 defines the ontological model that models the items contained in an eCatalog, and shows that useful information can be extracted from this structure. Then, chapter 4 explains why the ontology is not suitable for modeling the user's preference profiles, and instead, proposes to model preference profiles by a set of rated items. Third, chapter 5 defines in details the inference mechanism used by ontology filtering for inferring missing preferences. To overcome the problem of the ontology construction, chapter 6 proposes the idea of learning a set of taxonomies from past users' experience using distance based clustering algorithms, which are then transformed into ontologies. Based on the user's preferences, another algorithm is introduced that can select the best ontology from the set of learnt ones. Finally, chapter 8 contains experimental results that validate all of the introduced material.

Chapter 3

Modeling eCatalogs with ontologies

Despite the significant progress of recommender systems over the years, some profound problems remain. Chapter 2 highlighted the two main reasons for these problems: incomplete user profile and inadequate model of the items in the eCatalogs.

In this work, it is believed that the user's preferences follow some explicit or implicit ontology. Such ontology adds a structure to the items, which can be used to constrain the space of items a user will like. In this chapter, the idea of adding a hierarchical structure to eCatalogs is proposed in order to better extract the relationships between items. Furthermore, the attributes making an item can be implicitly hidden in the structure, which reduces the cognitive load on the user. Finally, given this structure, Chapter 5 shows that it is possible to infer missing information by transferring the information from one concept to another one. This novel inference process allows the system to estimate the missing user's preferences, which leads to a simpler and lighter elicitation process.

3.1 Introduction

The underlying model used to represent the items in the eCatalog is fundamental, and will influence not only the accuracy of the recommendations, but also the preference elicitation process. However, as shown in Figure 3.1, this model varies from one recommender technique to another.

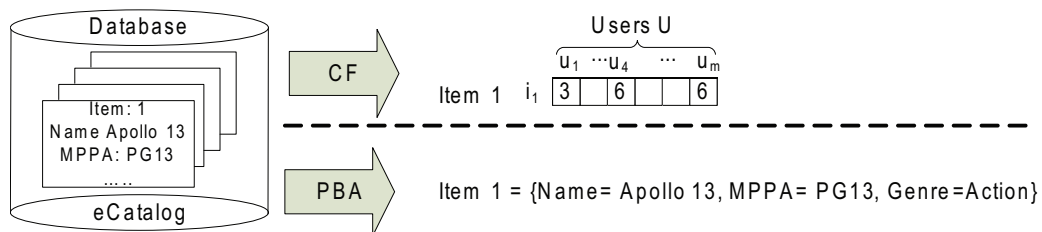


Figure 3.1: The models used to represent the items of an eCatalog by collaborative filtering and the preference based approach.

In collaborative filtering, items are modeled by a vector of users' ratings with an identifier, and the reasoning is done over basic pairwise similarities between pairs of these vectors. On the other hand, the preference based approach uses the attribute model, where each item is defined by a set of well defined attribute-value pairs.

Collaborative filtering's simple representation allows a user to simply state her preferences by rating a set of items. As explained in Section 2.2.2, the rating based approach is one of the simplest forms of preference elicitation that requires low cognitive effort from the user. Thus, it makes the recommender system very easy to use for novice users, and requires very little knowledge. Unfortunately, the lack of proper model leads to an unconstrained search space. As a consequence, collaborative filtering needs to elicit many ratings in order to locate the like-minded group of users.

Conversely, when the preferences are correctly elicited, the attribute model of the preference based approach guarantees to find the optimal solution. However, eliciting the user's preferences is a tedious process, which requires asking many value elicitation questions in order to build the appropriate utility function and weight of each attribute. Thus, this model requires the user to have a good domain knowledge, which limits this model to medium to high risk domain application. Another drawback of this approach is the rigidity of the attributes. This limits the evolution of the eCatalogs and makes it impossible to compare heterogenous items.

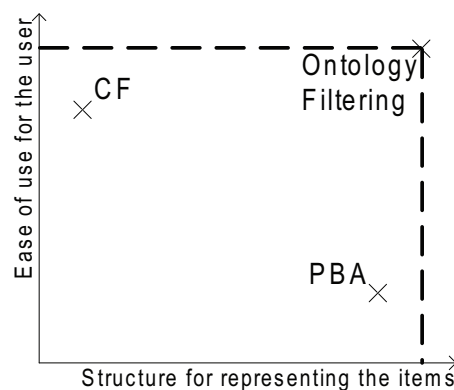


Figure 3.2: Graph that looks at two dimensions in a model: the ease of use for the user and the structure used for representing the items.

Current models fail to correctly model the contents of eCatalogs. To briefly summarize, collaborative filtering does not have a proper model for representing the items, while the preference based approach is too constraining for user in an eCommerce environment. This chapter proposes to model the items of an eCatalog by an *ontology*. The ontology is a hierarchical structure where a node represents a set of items, and an edge between two nodes models the inheritance relationship. The use of an ontology as a model has the following advantages:

1. *Ease of use for the user* - As for collaborative filtering, the recommender system uses rating based questions to elicit the user's preferences. However, instead of repeatedly asking questions to a user as CF does, the ontology model has the ability to infer missing preferences.

2. *Better structure for representing the items* - The ontology can also represent items by a set of attribute-value pairs, where each pair is modeled by an edge in the ontology. Informally, this can be seen as a decision tree, where the decision is replaced by the possible attribute-value pair. But the ontology can also model complex relationships between pairs of items. Another advantage of an ontology is the fact that features can be kept implicit in the model, which reduces its complexity. This dissertation acknowledges the fact that ontologies are very hard to construct and costly to maintain. To make the ontological model usable in eCommerce environment, algorithms are defined that generate the model automatically from raw data, and without any human interventions.

3.1.1 Ontology vs taxonomy

In this dissertation, the ontology structure is fundamental for representing the item, and for inferring missing user's preferences. One question that arises is, whether the reasoning structure is an ontology or rather a simple taxonomy?

Traditionally, a taxonomy is defined as a collection of entities that are organized into a hierarchical structure. With the rise of the semantic web, there are growing discussions about what is an ontology, and its differences from a taxonomy are often blurred. In its abstract philosophical notion, the online dictionary Merriam-Webster defines¹ an ontology as *a branch of metaphysics concerned with the nature and relations of being or a particular theory about the nature of being or the kinds of things that have existence*. In computer science, a commonly agreed definition is the one proposed by [Gruber, 1993] that states that an ontology is *an explicit specification of a conceptualization of the real world*. However, this definition remains very abstract and can lead to many interpretations. For example, [McGuinness, 2002] uses taxonomy interchangeably with simple ontology.

Often, an ontology will contain a subclass-based taxonomic hierarchy. But what differentiates an ontology from a taxonomy is the extra properties and tools that are added to the taxonomy in order to interpret the information that it contains. For example, value restriction property is a common property used in ontology. Reasoning languages like DAML-OIL² are also widely used in the semantic web to reason over the content of the ontology.

In this work, unsupervised learning algorithms are used to learn the hierarchical structures of the ontology. Two additional properties are added to each concept: the *score* and the *a-priori score*. The former property represents how much a specific user likes a given concept, while the latter is the probability of that concept being liked based on its location in the ontology. These two properties allow the recommender system to construct a full preference profile from an incomplete one. As a consequence, the learnt hierarchical structure altogether with the two properties form an ontology with a hierarchical structure. Notice that unless stated otherwise, and to simplify the writing of this chapter, this thesis uses the term ontology to denote an ontology with an inheritance hierarchical structure.

¹<http://www.m-w.com/dictionary/ontology>

²www.daml.org

3.2 Usage of ontologies in the Web

With the emergence of the semantic web and the growing number of heterogenous data sources, the benefits of ontologies are becoming widely accepted. Moreover, its domain of application is widening every day, ranging from traditional word sense disambiguation to search of biological macromolecules such as DNA and proteins.

Initially, ontologies were used in an attempt to define all the concepts within a specific domain and their relationships. An example of a popular ontology is the *WordNet* ontology [Miller et al., 1993], which models the lexical knowledge of a native English speaker. Information in WordNet is organized around lexical groupings called synsets and semantic pointers. Informally, a synset represents a set of synonym words, while a semantic pointer models the semantic relationships between two synsets. E-commerce sites such as Amazon.com or Yahoo also use simple taxonomies to classify their products. More recently, biologists and computer scientists have developed an ontology named *GeneOntology* [GO, 2000], which models biological processes, molecular functions and cellular components of genes.

Nowadays, the usage of ontologies goes far beyond domain specification. A very promising direction for ontologies is the Semantic Web, and more specially *semantic search* [Guha et al., 2003], where the structure of the ontology has been successfully be used to find documents [Davies and Weeks, 2004], pictures [Janecek et al., 2005], and even jobs [Bradley et al., 2000]. Semantic search is in fact an information retrieval application, where semantic knowledge captured by an ontology is used to enrich the available vocabulary. In comparison, traditional information retrieval applications use the Vector Space Model (VSM, [Frakes and Baeza-Yates, 1992]) to represent the set of possible items, and input query into a common vector space in order to compute the similarity between them. Unfortunately, if no document contains any of the input keywords, the VSM approach fails to find any relevant documents. To overcome this problem, semantic search uses domain ontologies to explore concepts similar to the stated keywords in order to build smarter search engines.

[Middleton et al., 2004] introduced the idea of representing user's preference profiles in ontological terms for recommending research papers. This idea was then extended by [Ziegler et al., 2004] for recommending items. Unfortunately, all these authors assume the existence of a predefined ontology that models the entire eCatalog. For example, Middleton et al. use the dmoz open directory project classification tree, while Ziegler et al. borrowed the Amazon.com taxonomy. However, there are two reasons why it is unrealistic to assume the existence of such ontology in an eCommerce environment. First, the items in eCatalogs are very volatile and constantly change. For example, new items with new features are added daily, while old items are removed as they become redundant. Second, it is very expensive and time consuming to build and maintain an ontology. This explains why only major online retailers such as Amazon.com or Yahoo can afford such ontology.

As for [Middleton et al., 2004], this dissertation uses an ontology to model the content of the eCatalog. However, there are four major differences that differentiate it from previous work. First, the existence of an ontology is not assumed. If none exists, then unsupervised learning algorithms are used to learn a set of ontologies. Two additional properties are added to each concept: the *score* and the *a-priori score*. Moreover, we show

in Section 5.3 that these two properties, along with the user's preferences, can be used to successfully infer new information. Third, ontology filtering do not restrict the structure of the ontology to tree, but takes advantage of more complex one such as directed acyclic graph. Finally, the user's preference profile is represented by a compact set of concepts, where each concept models an item a user has rated. Note that [Middleton et al., 2004] and [Ziegler et al., 2004] model the user's preference profile by a vector of concepts of size equal to the number of concepts in the ontology. As a typical ontology contains over 10'000 concepts (WordNet 2.0 has over 117'000 concepts), the vector of concepts approach will obviously not scale well in an eCommerce environment.

3.3 The multi-hierarchical structure of an ontology

Ontology filtering adds an ontology to eCatalogs in order to express the relationships between the items. This dissertation sees an ontology as a semi-balanced multi-hierarchical structure, where a node represents a primitive concept, and an edge models the binary specialization relation (*isa*) between two concepts.

The multi-hierarchical structure implies that a concept can have more than one parent. When all the concepts have at most one parent, then the structure can be considered as a tree as this dissertation only considers inheritance relations. However, the inheritance multi-hierarchical structure is a *directed acyclic graph*, DAG, if at least one concept has more than one parent. A directed acyclic graph is a directed graph with no directed cycles; that is, for any vertex v , there is no nonempty directed path that starts and ends on v . Moreover, DAGs can be considered to be a generalization of trees in which certain subtrees can be shared by different parts of the tree. Figure 3.3 illustrates two possible structures for a transport ontology, where 3.3(a) contains a simple tree structure and 3.3(b) has a DAG structure. In Figure 3.3(b), the Amphibious concept represents amphibious vehicles that have the capability of traveling on both land and sea. These types of vehicles are becoming increasingly popular tourist attractions in town that have rivers such as London and Boston.

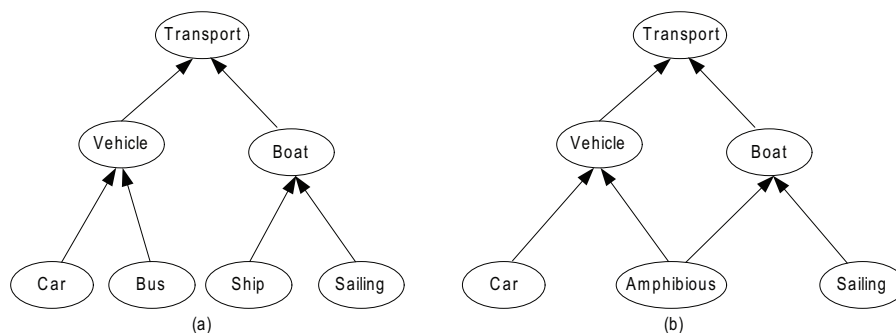


Figure 3.3: Two ontologies with a tree (a) and DAG (b) structure.

With this kind of ontology, an item will be instance of one or more concepts, and the edges will represent implicit or explicit features. Each concept can have a set of sub-concepts known as the *descendants*, but not all instances of a concept must belong to a sub-concept. Consequently, items in the different sub-concepts are distinguished by differences in certain features. However, these are usually not made explicit in the ontology.

Concretely, we see a feature as a restriction on a property or a combination of properties that differentiates a concept from its parent.

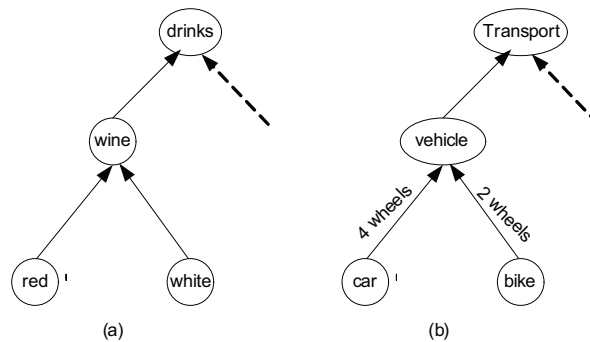


Figure 3.4: Two ontologies with implicit (a) and explicit (b) features.

For example, Figure 3.4(a) shows a possible ontology for drinks. The concepts *red* and *white* respectively represent the instance of red and white wines. In this situation, the features are implicit, as the concepts red and white are distinguished by a combination of features which include color and also certain aspects of taste. Notice that in Figure 3.4(b), the concepts *car* and *bike* are distinguished by the explicit feature *#wheels*.

When experts construct ontologies, they usually ensure that ontologies are more or less balanced. In an eCommerce environment, ontologies continuously evolve, which makes it difficult for experts to maintain them balanced. In this work, we use the topology of the ontology to infer missing knowledge. As a consequence, it is important to maintain a nearly-balanced structure in order to correctly estimate the preference of a concept. To achieve this objective, this dissertation proposes to learn the ontologies automatically from hierarchical clustering algorithms instead of using predefined ones. Clustering algorithms allow the ontology to be recomputed dynamically, while maintaining (to a certain extent) the same number of instances in each leaf cluster.

3.3.1 Properties of the score of a concept

The main objective of a recommender system is to help a user find the items that meet her preferences. As a consequence, the user's behavior, along with her preferences, have to be integrated in the core of the recommender system. When a user said that she liked an item, this usually means that she liked some, or all of the features making that item. For example, a user could like the James Bond movie "Tomorrow never dies" because it is an action movie that contains the actor "Pierce Brosnan". In this case, both the genre and the actor are the features the user is interested in. In some situations, these features are not always as explicit, and sometimes users are not even aware of them. Take for example a bottle of wine. Most people will usually have a preference, either red or white wine; but only few people will be able to clearly say why. This is because the features (like taste) are usually more implicit than simple red or white color.

One of the most crucial behaviors a user has (and which has been ignored by nearly all recommender systems) is the *risk-aversion*. Moreover, it is commonly agreed that most people tend to act in a risk averse manner in their daily life. This means that, under uncer-

tainty, people do not maximize the expected profit but prefer sure rewards. Consider for example the (real) situation of driving behaviors on Swiss highways. In Switzerland, the speed limit on highways is 120km/h, which corresponds to the average most Swiss persons do. Last year, the canton de Vaud installed brand new speed camera every five kilometers on the highway linking Lausanne to Geneva. As a result, and even if the threshold for these speed camera is set to 128km/h, most people pass them at no more than 115km/h. Moreover, people significantly reduces their speed on the approach of these cameras by fear of being caught, even when driving well under the speed limit.

Following these two fundamental observations, the *score* of a concept is defined as follows.

Definition 3.1 *The score of a concept c , $S(c)$, is a real value function defined in the interval $[0..1]$, which satisfies the following properties:*

- *A1: the score depends on the features of a concept.*
- *A2: each feature contributes independently to the score.*
- *A3: features that are unknown make no contribution to the score.*

Assumption A1 is very intuitive and reflects the fact that a concept is modeled by a set of features. Thus, the score will only be influenced by the features making up the concept. It is also the basis of multi-attribute decision theory [Keeney and Raiffa, 1993], where the utility of an item depends on the preference value of the attributes making that item. Thus, all instances of the same concept will have the same score, as they share the same features.

The second assumption eliminates the inter-dependence between the features, and allows the score to be modeled as the sum of the scores assigned to each feature. In the multi-attribute utility theory, an even stronger assumption (the mutual preferential independence) is used to build an additive value function for an item [Keeney and Raiffa, 1993] [Payne et al., 1988]. Independence is a strong assumption, but it is still more accurate than the assumption that all the features correlate positively to the score.

The third assumption is crucial and reflects the observation that users are risk averse. For example, if the score models the price that a user is willing to pay for an item, it is rational for users to adopt this pessimistic view, since one would not normally be willing to pay for features that have not been explicitly provided or which are disliked. Thus, the score attached to a concept can be seen as a lower bound on the score that items belonging to that concept might have.

More generally, some analogy can be made between the score function and the *lower prevision* [Walley, 1996]. The lower prevision of a gamble X is a real number, which is interpreted as the highest price a user is willing to pay for X . In fact, the score of a concept c corresponds to a strict lower bound of the prevision for selecting any instance of c . Preferences can be reasonably modeled as scores, but scores could also model other properties.

In the literature [Rissland, 2006], real life concepts are usually referred to as *messy concepts*, which posses the following characteristics:

Nonstationary concepts change over time. These changes can be sudden in the case of shift in technology, or more gradual. Real-world concepts exhibit tremendous changes, as the world we live in constantly evolves. Even mathematical concepts change when counterexamples are found.

Open-textured concepts do not have hard boundaries offering black-and-white distinctions between positive and negative instances. The world is full of such concepts. Take for example the law that states that *vehicles* are not allowed to travel on pavements. But what counts as a vehicle? Most people would agree that a car is such a vehicle. But what about a Segway³? Current Swiss legislation forbids the usage of such vehicles on the roads as they are not *big* enough, and on pavements as they are motorized.

Nonconvex concepts have exceptions or holes. These are negative examples that reside in the concept interior, where only positive one ought to be. Formally, exceptions in concepts can occur if a negative example resides in the concept's interior.

Ontology filtering uses ontologies to model eCatalogs, where a concept represents a set of items, each having a possible rating assigned by the user. To allow the ontology to evolve and deal with nonstationary and open-textured concepts, ontology filtering uses clustering algorithms that can (re)generate the ontologies in a regular time interval. To consider nonconvex concepts, this dissertation proposes to compute the score of a concept c for a user u as follows.

$$S(c) = \frac{\sum_{i \in L} \hat{R}_{u,i}}{|L|} \quad (3.1)$$

where L is the set items that have been rated by the user u that are instanced of concept c , and $\hat{R}_{u,i}$ is the normalized rating of item i assigned by user u . The average computation allows to compensate the fact that some items (exceptions) have been incorrectly classified.

3.3.2 The a-priori score of a concept

An ontology is usually designed in such a way that its topology and structure reflects the information contained within and between the concepts. A major ingredient of ontology filtering is the computation of the *a-priori score* of a concept c , $APS(c)$, which captures this information. The APS models the expected score of each concept for an average user, but without using any user information. It is not used as a prediction of actual scores, but only to estimate constants (α and β) that determine how actual users scores propagate through the ontology.

As no information about a specific user is available, ontology filtering assumes that all concepts have a score that is uniformly distributed between 0 and 1. This is often a reasonable assumption as each concept exists to satisfy the desire of some group of people. Thus, the probability that the score of a concept c is superior to the threshold x , $P(S(c) > x)$, is

³<http://www.segway.com/>

equal to $1 - x$. However, this probability ignores the fact that concepts can have descendants. Furthermore, the score function is by definition pessimistic (A3), which implies that the score should be a lower bound of the score of its instances and the score of its descendants. Therefore, the probability that the score of any concept c is superior to a threshold x is equal to $(1 - x)^{n+1}$, where n is the number of descendants of c . Note that we count all descendants of c and c itself, to account for the fact that each concept may have instances that do not belong to any sub-concept. Thus, to like a concept c , a user must like all of instances of its descendants, as well as all the instances of c . Take for example the ontology contained in Figure 3.5, where concept c has four descendants. For a user to like concept c , she must like all the instances found in concepts d, e, f, g , and all the instances in c .

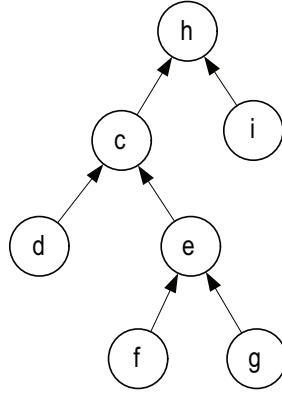


Figure 3.5: A simple ontology to illustrate the a-priori score of the concept c .

Thus, the probability distribution of the score for a concept c is $P(S(c) \leq x) = 1 - (1 - x)^{n+1}$, with the following density function:

$$f_c(x) = \frac{d}{dx} (1 - (1 - x)^{n+1}) = (n + 1) \cdot (1 - x)^n \quad (3.2)$$

As a consequence, the expected lowest bound of the score of a concept c , $E(S(c))$, can be obtained by integration of the density function as follows.

$$\begin{aligned} E(S(c)) &= (n + 1) \int_0^1 x(1 - x)^n dx \\ &= (n + 1) \left[\underbrace{\left| -\frac{x(1 - x)^{n+1}}{n + 1} \right|_0^1}_0 + \underbrace{\int_0^1 \frac{(1 - x)^{n+1}}{n + 1}}_{1/((n+1)(n+2))} \right] \\ &= \frac{1}{n + 2} \end{aligned} \quad (3.3)$$

Equation 3.3 shows that the expected score of a concept c will be inversely proportional to the number of its descendants + 2. Following this, we define the a-priori score of a concept c with n descendants as:

$$APS(c) = \frac{1}{n + 2} \quad (3.4)$$

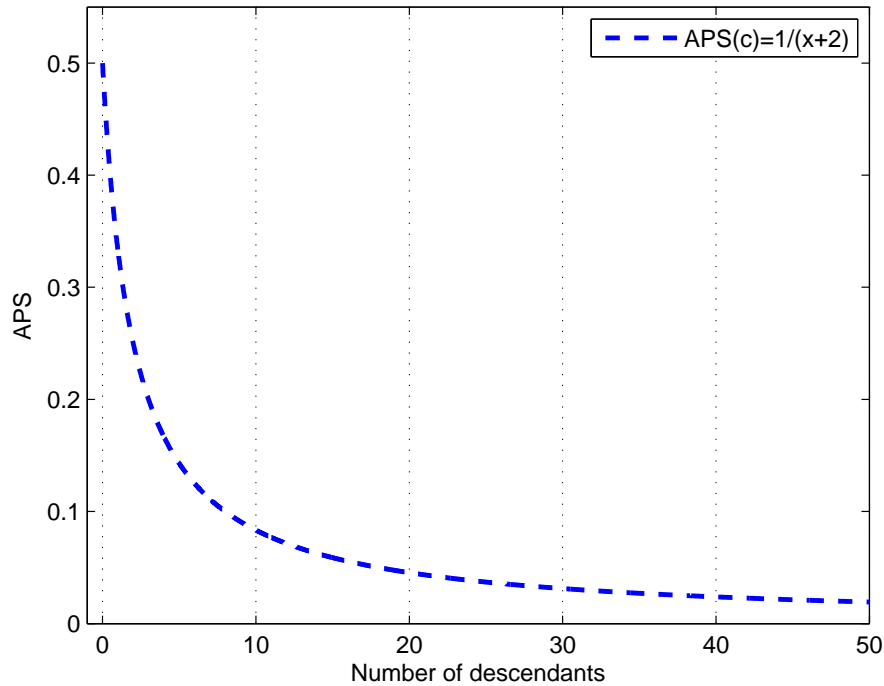


Figure 3.6: The behavior of the a-priori score of a concept c when the number of descendants varies from 0 to 50. The x-axis is the number of descendants of the concept c , while the y-axis measures the APS .

The a-priori score defined in Equation 3.4 implies that the leaves of the ontology will have an APS equal to $1/2$, which is equal to the mean of a uniform distribution between 0 and 1. Conversely, the lowest values will be found at the root. This means that when we travel up the ontology, concepts become more generalized, and therefore the APS of concept decreases. As shown in Figure 3.6, the difference in APS between two concepts is much smaller when we travel up the ontology. This is a desired property that reflects the fact that concepts become more generalized due to the increasing number of descendants, and that features of specific concepts are more important to the user than features of generalized concepts.

To better understand these properties, consider the snapshot of the WordNet ontology contained in Figure 3.7. The *red wine* concept is a leaf concept, which implies that its APS is equal to $1/2$. The concept *wine* has two descendants: *red wine* and *white wine*. Thus, these two descendants force the APS of concept *wine* to $1/4$. Under the assumption that the ontology is balanced, the APS of the *alcohol* concept is equal to $1/8$. Note how the difference in APS decreases when going up the ontology. For example, the difference is $1/4$ between concepts *red wine* and *wine*, while only $1/8$ between concepts *wine* and *alcohol*. This models the fact that features of specific concepts are more important to the user than features of generalized concepts. Take for example a user who states as a preference that she likes red wines. This usually implies that the edge between concepts *Red Wine* and

Wine is the most important to the user, as it is the one that distinguishes red wines from white wines. Inversely, the edge between concepts *Substance* and *Entity* has the lowest importance to the user as it is too abstract. Most users who express preferences on a bottle of red wine will not even be aware that the *Red Wine* concept is a descendant of the *Entity* concept.

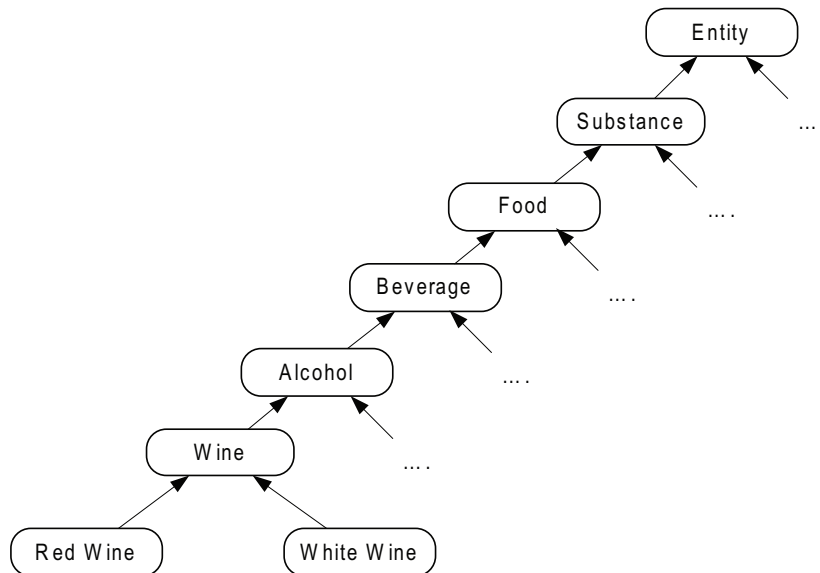


Figure 3.7: A snapshot of the WordNet Ontology for red wine.

To illustrate the computation of the a-priori score on an entire ontology, consider the simple ontology λ shown in Figure 3.8(a). First, the number of descendants of each concept n_c is computed. Then, Equation (3.4) is applied to compute the APS of each concept in λ .

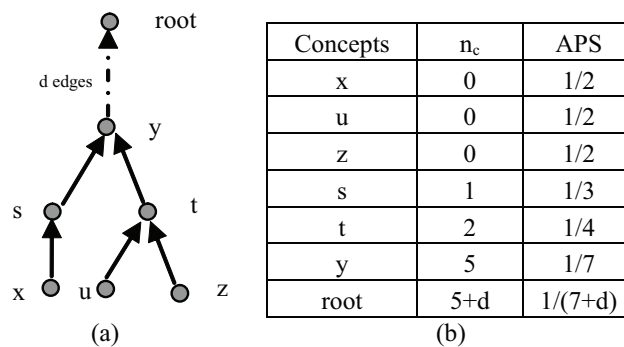


Figure 3.8: (a) a simple ontology λ and its APSs computed using Equation (3.4)(b).

3.3.3 Analogy with existing work

Resnik also uses the topology of an ontology to compute the *information content* of a concept, (IC, [Resnik, 1995]). Following the information theory, Resnik defines the information content of a concept c as the negative log likelihood of c occurring, $-\log P(c)$, where $P(c)$ is the probability of encountering an instance of concept c . The information content

of a concept is then used to compute the similarities between pairs of concept. Note that similarity metrics are defined later in Chapter 5.4.1.

The a-priori score shares some similarities with the information content. For example, the difference in both APS and IC decreases between two concepts, when we travel up the ontology. However, some profound differences exist that are consequences of the way the APS and IC are computed.

First, Figure 3.9 clearly shows that both functions do not have the same range of values. The information content varies from 0 to 1 when normalized, while the a-priori score is always included in the open interval $(0,0.5]$. The information content measure is clearly not applicable in a recommendation context as the probability of liking all leaf concepts is very unlikely be one. Similarly, the probability of a user liking the root concept cannot be 0 either, as it would mean that no user could like all the concepts of the ontology. Some analogy can be made with throwing a dice. The probability of throwing a dice once and getting a value superior to three is $1/2$. However, the probability of throwing a dice 50 times and still getting a value superior to three each time is $(1/2)^{50}$, but not 0.

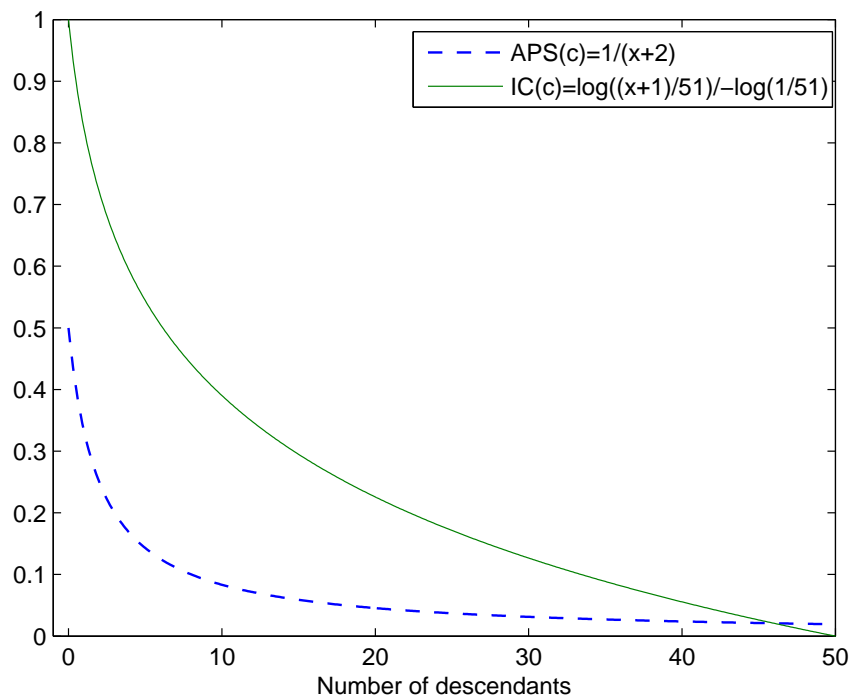


Figure 3.9: Comparison of the APS and IC for concepts that have between 0 and 50 descendants. The concepts with 50 descendants is considered as the root of the ontology.

Second, as the information content is computed using the probability of encountering a concept c , this implies that whenever a new concept is added to the ontology, or when existing concepts are merged/split, the probability of all the concepts need to be recomputed. Obviously, this seriously limits the evolution of the ontology and its integration with others. The a-priori score is more robust to this problem as the APS of a concept c is computed using only the number of descendants of c . For example, imagine the situation

where a new concept d is added to an existing ontology. Now, take any concept c from the ontology. If d is not a descendant of concept c , then $APS(c)$ will remain unchanged; even if d is a parent of c . However, when d is a descendant of c , then the $APS(c)$ will decrease. As a consequence, the APS will be more sensitive to the number of descendants, which could potentially lead to problems if the ontology is not balanced. If the ontology is not semi-balanced, then concepts at the same height may have very different APS. As the inference process of ontology filtering uses the APS of a concept, this phenomenon introduces a bias towards some concepts. This work acknowledges this problem, and this is another reason why this dissertation proposes to learn the ontologies using hierarchical clustering algorithms (which try to have a balanced structure, while having the same number of instances in each leaf concept). Finally, experimental results in Section 8.1 tend to show that the APS along with the ontology filtering's inference mechanism is better than IC for simulating the way a user perceives distances between pairs of concepts in an ontology.

3.3.4 The ontology

Following the definitions of the score and a-priori score of a concept, an ontology in ontology filtering is formally defined as follows.

Definition 3.2 An ontology λ is a semi-balanced multi-hierarchical structure, where a node represents a primitive concept c , and an edge models the binary specialization relation (*isa*) between two concepts. Furthermore, each concept c contains the following properties:

1. $S(c)$ is the score of the concept c ,
2. $APS(c)$ is the a-priori score of the concept c .

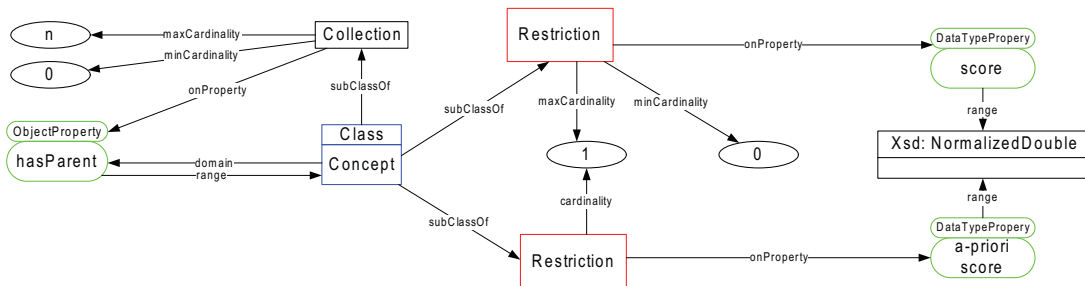


Figure 3.10: The ontology λ in DAML-OIL language.

Notice that unless stated otherwise, and to simplify the writing, we use the term ontology to denote an ontology with an inheritance semi-balanced hierarchical structure. To illustrate the definition of our ontology λ , Figure 3.10 shows a graphical representation in the DAML-OIL language. Given the definition of λ , the class *Concept* is made of three properties: *hasParent*, *score*, and *a-priori score*. The *hasParent* property models the fact that a concept can have a parent, while the *daml:collection* parse type with the maximum cardinality set to n limits the number of parents to a finite number. This property allows to model our inheritance hierarchical structure, where the features remain implicit in the

hierarchical model. Note that in practice, the number of parents n tends to be small, and is on average less than two. For example, concept in the WordNet ontology have on average 1.03 parents. Obviously, the score and a-priori score properties respectively model the score and a-priori score of the concept. Notice that the property score has respectively a minimum and maximum cardinality of 0 and 1. When the cardinality is set to 0, then this means that no preferences were stated by the user. Inversely, a cardinality of 1 means that the user has expressed preferences on the concept.

Given the ontology defined in Definition 3.2, the items of an eCatalog can be modeled by associating them to some concepts of the ontology. As a consequence, an item will be instance of one or more concepts, and the edges will represent implicit or explicit features

3.4 The eCatalog ontological model

To conclude this chapter, the model used to represent the items of an eCatalog is illustrated in Figure 3.11, and formally defined as follows.

Definition 3.3 *The eCatalog ontological model, $eCOM$, is an ontology λ as defined in definition 3.2, where each item of the eCatalog is an instance of a primitive concept in λ .*

To construct this ontological model, this dissertation proposes Algorithm 1 that makes use of a predefined structure modeling the eCatalog. For now, it is assumed that such structure is made available to the system, but Chapter 6 will focus on the problem of creating it. Formally, the algorithm is as follows. First, the eCatalog ontological model, $eCOM$, is initialized with the semi-balances multi-hierarchical structure $eDAG$. For each concept in $eCOM$, step 3 computes its a-priori score using Equation 3.4, while step 4 sets its score to 0. Recall that the a-priori score of a concept is independent of a user, and only considers the information contained in the concept as a function of its number of descendants. However, the score property is user specific and its value cannot be known without any preferences from the user. Once all the APSs and scores have been computed, the algorithm terminates by returning the eCatalog ontological model.

Algorithm 1: *Generating the eCatalog ontological model.*

Inputs: an inheritance semi-balances multi-hierarchical structure $eDAG$ modeling the eCatalog

Outputs: An eCatalog ontological model $eCOM$.

Functions: $aprioriscore(c)$ computes the a-priori score of a concept c using Equation 3.4.

```

1  $eCOM \leftarrow eDAG$ 
2 forall  $c \in eCOM$  do
3   |  $APS(c) \leftarrow aprioriscore(c)$ 
4   |  $S(c) \leftarrow 0$ 
   end
5 return  $eCOM$ 

```

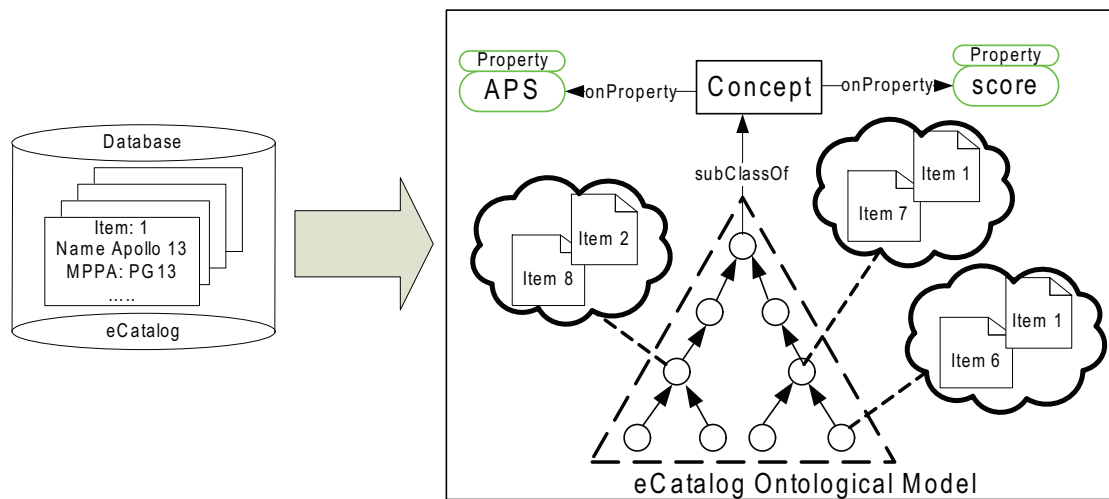


Figure 3.11: Illustration of the eCatalog ontological model generated with Algorithm 1.

Chapter 4

Modeling the users' preference profiles

This chapter presents the model used to represent the user's preference profile. Contrary to the previous chapter, the user's preference profile will not be modeled by an ontology, but by a set composed of the items a user has previously rated.

*Things should be made as simple as possible, but not any simpler.
Albert Einstein 1879-1955*

4.1 Introduction

In the previous chapter, the idea of modeling eCatalogs with an ontology was proposed. Along this ontology, two properties were defined with each concept that differentiate this structure model from a simple taxonomy: the a-priori score and the score. While the a-priori score characterizes how much an average user likes a given concept, the score however measures how much a specific user likes it.

A naive user model would consist at modeling the user's preferences by some concepts extracted from the ontology, where the user will set her preferences by setting the score of some concepts. For example, a system could elicit the user's preferences by asking her, through rating based question, the score of a predefined set of concepts.

Nevertheless, this chapter does not propose to use the ontology to directly model the users' preference profiles. Instead, this dissertation models a user preference profile by a set composed of the items a user has previously rated. There are two reasons for not using an ontology:

1. *Ontology lifetime* - As previously argued, the content of eCatalogs continuously changes as novel items are added daily, while redundant ones are removed. Thus, the ontology modeling such domain will become outdated very soon, and will need to be recreated. Moreover, Chapter 6 proposes an algorithm that learns a set of ontologies automatically, so that ontologies can generated daily. Thus, the preference assigned to a concept will also be outdated, as the rated instances may not belong to the same concept. To fully understand the consequence of the problem, imagine a recommender system running on an eCommerce retailer with millions of users (such

as Amazon.com). If concepts were used as modeling objects, then this implies that each time the ontology changes, all the preferences of each concept of each user would have to be updated. Obviously, this becomes unfeasible with millions of user and items.

2. *Cognitive overload* will be the direct consequence if the recommender system directly asks the user to rate concepts. As concepts can represent more than one item, this could lead to ambiguity when edges model implicit features. Furthermore, the user will need to have constant knowledge of the meaning of each concept. This assumption clearly does not hold in the eCommerce environment, where anyone can come and query the system.

As a consequence, this chapter proposes to model the user's preference profile by a set of ratings that the user has assigned to items she has previously experienced.

4.2 Eliciting the user's preferences

Before modeling the user's preferences, the system needs to elicit them from the user. In low user involvement decision processes, this is expensive as a user is usually not willing to spend much effort expressing her preferences. As introduced in Chapter 2.2.2, [Smyth and McGinty, 2003] have identified four strategies for eliciting preferences from a user: value elicitation, rating based, critiquing, and preference based. To illustrate these strategies, imagine a recommender system who wants to recommend movie to users. Given an elicitation strategy, the recommender system elicits the user's preferences by asking the following questions:

- *Value elicitation* - Who is your favorite *actor*? Do you like *drama* movies?
- *Rating based*- How much do you like the Apollo 13 movies? (from 1 to 5).
- *Critiquing* - Given a 1 hour movie, do you want to see a *longer one*?
- *Preference-Based* - Which movie do you prefer: The Lion King or Apollo 13?

It is very difficult to decide which approach to use, as each one has its advantages and problems. In low user involvement decision processes, [Smyth and McGinty, 2003] suggested that the rating-based approach is one of the easiest ways of getting the user's preferences. Furthermore, ratings on items are becoming widely available as users are continuously asked to rate items they have either seen or bought. For example, when a user has bought an item on ebay.com¹, ebay will ask the user to evaluate the seller through a rating and additional comments. Similarly on YouTube², users can also rate videos they have seen. However, asking users to rate items they have experienced is not unique to web applications. For example, Microsoft Office asks a user to rate the help tips she received after facing a problem. In marketing, satisfaction forms also use ratings to evaluate the customer satisfaction about a product or services she received.

¹www.ebay.com

²www.youtube.com

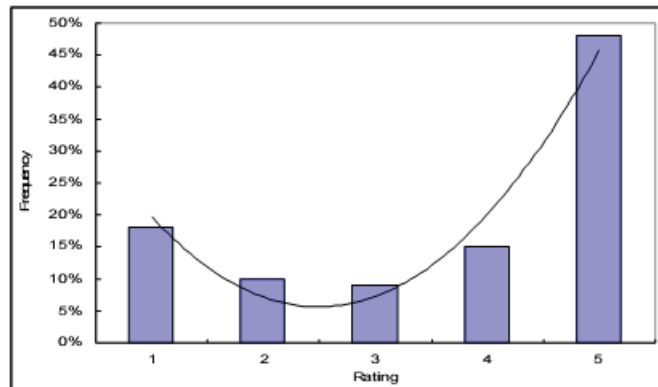


Figure 4.1: Distribution of the ratings on Amazon.com for the randomly picked CD "Mr A-Z". This figure has been extracted from [Hu et al., 2006].

Following this, this dissertation uses a variant of the rating based strategy to elicit the user's preferences. Instead of asking the user to rate a given number of items, we will ask the user to provide us with the ratings of at least 5 items of her choice. As we are interested in finding items the user will like, we ask the user to provide us with at least 70% of items he really liked, and at least 20% of items he strongly disliked. There are two main reasons why ratings are elicited this way. First, as shown later in this dissertation, missing scores are inferred from the closest concepts which have known user's scores. As information gets lost during the inference, it is very important to start from a concept that is very meaningful to the user. Another important aspect is that users tend to remember better items that they have really liked, or the one that they hated. As shown in Figure 4.1, [Hu et al., 2006] found similar polarization of rating behavior on extreme ratings for Amazon.com. As a consequence, it is easier for them to express preferences on these items rather than on items the system may have selected. Obviously, users will not always be able to say which items she really liked or strongly disliked. That is why a user can rate randomly selected items which are shown to her.

As with traditional recommender systems, this thesis uses a 5 stars rating scale to rate items. The value 1 means that the user strongly dislikes the item, while 5 means she loves it. The threshold of liking an item is set to 4, which means that any item with a rating less than four is considered as being disliked by the user.

4.3 The user's preference profile

In ontology filtering, the user's preference profile is defined as follows.

Definition 4.1 *The user's preference profile is a set of items that the user has previously rated. These ratings are defined on the interval $[1, 5]$, where the value 1 means that the user strongly disliked the item, while 5 means she loved it.*

Formally, the preference profile defined in Definition 4.1 is constructed using Algorithm 2. First, step 1 verifies if a user u has already a valid profile. If it has, then the two conditions in step 4 will return *False*, and the algorithm will simply return the user's preference profile stored in the database. Note that this step is essential as a user can

modify her preference profile at anytime, and thus theoretically, delete all of them. On the other hand, when the preference profile is incomplete, steps 4 to 11 elicit the missing information from u . As explained in the previous section, the elicitation algorithm asks for at least 5 ratings from the user, while assuring that at least 70% of the elicited items are really liked by the user, and at least 20% are really disliked. Thus, step 5 elicits the rating from the user, while step 4 assures that resulting preferences meet the above criteria. As the user is free to give a rating on any item of her choice, step 6 makes sure that the rated item belongs to the eCatalog ontological model. If the user's preference profile has changed (i.e.: the *flag* is *True*), then step 13 updates the version found in the database in order to maintain coherence and durability of the preference profile.

Algorithm 2: *Building and retrieving a user's preference profile for a user u .*

Inputs: A user u , and the eCatalog ontological model $eCOM$.

Outputs: A user's preference profile UPP .

Functions: $getUPPFromDB(u)$ retrieves the preference profile of user u if it exists, otherwise returns $\{\phi\}$; $elicit(u)$ asks the user to give one item with a rating and returns a tuple $\langle item, rating \rangle$; $ratioIsOk(UPP, 0.7, 0.2)$ checks whether the user's preference profile contains at least 70% of items u really liked, and at least 20% of items u really disliked; $updateUPPinDB(u, UPP)$ updates or copy the user's preference profile in the preference profiles database.

```

1  $UPP \leftarrow getUPPFromDB(u)$ 
2  $counter \leftarrow 5 - |UPP|$ 
3  $flag \leftarrow False$ 
4 while  $counter > 0$  or  $!ratioIsOk(UPP, 0.7, 0.2)$  do
5    $\langle item, rating \rangle \leftarrow elicit(u)$ 
6   if  $item \notin eCOM$  then
7     | Go back to step 4
   end
8   else
9     |  $UPP \leftarrow UPP \cup \langle item, rating \rangle$ 
10    |  $counter--$ 
   end
11   $flag \leftarrow True$ 
   end
12 if  $flag$  then
13   |  $updateUPPinDB(u, UPP)$ 
   end
14 return  $UPP$ 

```

Table 4.1 shows an example of user's preference profile that was obtained using Algorithm 2. As one can see, it contains exactly 80% of items that the user loved, and 20% that he hated. Note that among this set of preferences, the user loves all the movie except *Marry Poppins*, which is strongly disliked.

| Item | rating |
|---|--------|
| Lord of The Rings - Two Towers | 5 |
| Star Wars Episode II - Attack of the Clones | 5 |
| The Matrix Reloaded | 5 |
| The Lion King | 5 |
| Mary Poppins | 1 |

Table 4.1: Example of the user preference profile for user u .

To maintain the durability of the users' preference profiles, they are stored in a database system located in the data layer. As shown in Figure 4.2, once stored, a user preference profile can be updated by the following two processes:

1. *The recommendation process* contains two sub-processes that modify the user's profile: the preference elicitation and the preference feedback. When a new user connects to the system, the former builds the user's profile using the elicitation approach defined in the previous paragraph. On the other hand, the latter subprocess adds new preferences if the user decides to rate the recommended items.
2. *The manual preference update process* allows the user to update her preference profile at any time by adding, updating or deleting any rating she may have.

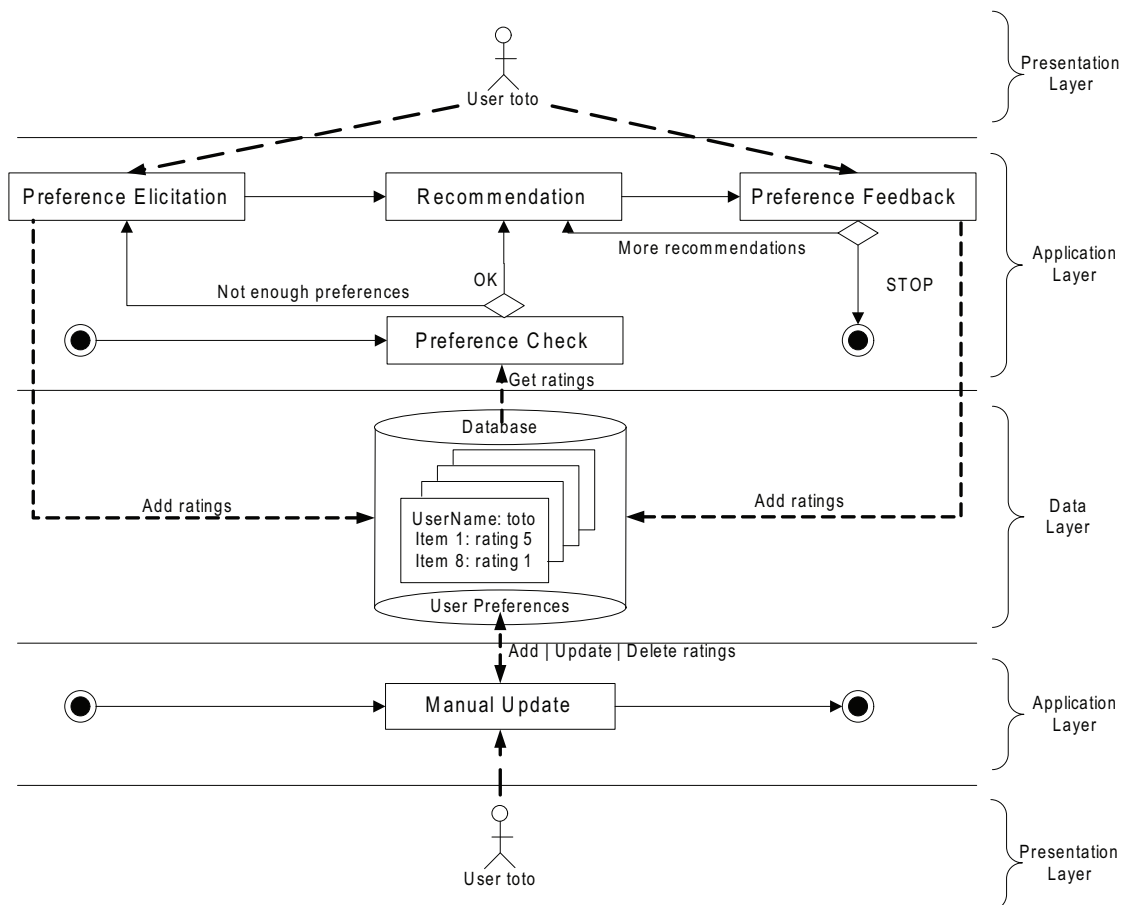


Figure 4.2: The two processes that can modify the user's preference profile.

Note that in practice, it is useful to remove outdated preferences as people's tastes tend to change over time. However, this dissertation does not consider this problem, but more information about this issue can be found in [Viappiani et al., 2002].

Chapter 5

Inferring missing user's preferences

The elicitation problem has been identified as one of the problems faced by recommender systems. One of the major contributions of this thesis is to propose the idea of inferring missing user's preferences instead of asking questions to the user.

This chapter defines in detail the inference mechanism that is used to infer missing preferences. As shown later in this chapter, the inference is done by transferring the score from the closest concept that contains some preference. Thus, this chapter starts by showing that the score of a concept can be transferred from one concept to another. It continues by defining a new similarity metric, which allows the system to compute the distance between any pair of concepts in the ontology. Then, an inference mechanism is defined that combines the transfer of scores and the similarity metric, along with an example. Finally, this chapter is concluded with the algorithm that selects the top- N items to be recommended to the user.

5.1 Introduction

Personalized recommender systems recommend items to a user based on her preferences. Moreover, it is commonly agreed that the more preferences the system has about a user, the more accurate the predictions will be. In an ideal world, users would answer any elicitation question asked to them, which would allow the system to build complete and accurate preference profiles.

Unfortunately, recommender systems are not deployed in an ideal world, where users' preferences are abundant and missing preferences can simply be elicited. Furthermore, this thesis focuses in low risk domains, where user's preferences are very hard to obtain. Two main reasons explain why preferences are not abundant. First, users have little knowledge about the domain, which makes it very hard to express preferences. Second, as the cost of failure is very low, users are not ready to spend hours expressing their preferences.

As explained in Chapter 2.5.1, many authors have come with solutions to overcome the first problem. For example, [Burke et al., 1997] proposed the idea of critiquing the current proposed recommendations through predefined critiques. These critiques are very easy to understand from the user point of view, and makes it easy for the user to express more preferences with little domain knowledge. As mentioned previously, the two limitations

of this work were the fact that the critiques needed to be predefined by hand, and were constant through out all the recommendation cycles. To overcome these two problems, [Reilly et al., 2004] proposed dynamic critiquing, which allows the compound critiques to be generated and selected during each recommendation cycle. Later, [Viappiani et al., 2006] proposed another method for stimulating the user to express more preferences by showing her suggestions that could become optimal if the user states extra preferences. All these approaches allow users to gain more knowledge about the domain, which stimulates the user to express more preferences but also increases the confidence in the recommendations. However, they remain limited when users are unwilling to give more preferences.

This thesis proposes a totally different approach. Instead of asking the user for more preferences, the recommender system tries to *estimate* them by using known user's preferences. The fundamental idea is to transfer existing preferences, captured as concept's scores, from one concept to another. The proposed inference mechanism is a two steps process. First, the closest concept with preferences to a given concept is identified using a novel similarity function called *OSS*. Then, the preference of this concept is propagated to the other concept through the lowest common ancestor between both concepts.

This chapter starts by merging previous chapters, and shows how the user's preference profile can be translated into an ontological profile, which is made of concepts and score values. Then, the inference mechanism that allows the transfer of scores between any pairs of concepts in an ontology is defined. Given this mechanism, it is showed that a similarity function can be derived from it. Moreover, experimental results in Chapter 8 show that this new similarity function outperformed current similarity metrics on the WordNet and GeneOntology. These two techniques are then combined together to allow ontology filtering to transfer the score of the closest concepts with preferences. Finally, this chapter finishes by giving the algorithm that selects the items to be recommended to the user.

5.2 From user's profile to user's ontological profile

Chapter 3 uses the eCatalog ontological model to model the items of an eCatalog, and also introduced the score and a-priori score associated to each concept of the ontology. However, in Chapter 4, it was argued against the use of an ontology to model the users' preference profiles, but in favor of a simpler model composed of rated items. Recall that the two main reasons for not using the ontology was its lifetime and the cognitive overload when reasoning over concepts.

Ontology Filtering uses an ontology to infer missing preferences. As a consequence, the user's preference profile needs to be integrated to the ontology. This integration can be performed using Algorithm 3, where $R_{u,item}$ is the rating assigned by user u on item $item$, $c.ratings$ is a set attached to concept c that contains all u 's ratings assigned to instances of c , and $rating$ is a normalized rating in the interval $[0, 1]$.

In words, Algorithm 3 works in the following three steps:

1. *Initialization* - In step 1, the algorithm starts by creating an empty set UOP that will represent the user's ontological preference profile. This set will contain some concepts from the ontology of the eCOM, where the instances of the selected concepts correspond to the previously rated items.

Algorithm 3: *Building the user's ontological profile for a user u .*

Inputs: The user's u preference profile UPP , composed of rated items; and the eCatalog ontological model $eCOM$;

Outputs: The user's u ontological profile UOP , which is a set of concepts from $eCOM$ with an associated score

```

1  $UOP \leftarrow \{\phi\}$ 
2 forall  $item \in UPP$  do
3    $c \leftarrow \text{concept} | \text{concept} \in eCOM \wedge item \in \text{concept}$ 
4   if  $c \in UOP$  then
5      $c.ratings \leftarrow c.ratings \cup R_{u,item}$ 
6   end
7   else
8      $c.ratings \leftarrow \{R_{u,item}\}$ 
9      $UOP \leftarrow UOP \cup c$ 
10  end
11 end
12 forall  $c \in UOP$  do
13    $S(c) = \frac{\sum_{rating \in c.ratings} \widehat{rating}}{|c.ratings|}$  (Equation 3.1)
14 end
15 return  $UOP$ 

```

2. *Concept Extraction* - For each rated item $item$ in the user preference profile, step 3 extracts the concept c that has an instance modeling $item$. If the concept is already present in the user's ontological profile, then the rating assigned to the item is simply added to the set of ratings of that concept. Otherwise, the set of ratings of that concept is initialized to the item's ratings, and the concept is added to the user's ontological profile.

3. *Score Computation* - Using Equation 3.1 defined in Section 3.3.1, the score of each concept in the user's ontological profile is computed by averaging the normalized user's ratings associated to that concept.

Given Algorithm 3, the user's ontological profile is defined as follows.

Definition 5.1 *The user's ontological profile, UOP , is defined as a set of concepts with their associated scores. The UOP is constructed from the user's preference profile using the eCatalog ontological model and Algorithm 3.*

Figure 5.1 illustrates definition 5.1 on a user $toto$, who has the same preference profile as the one shown in Table 4.1. Using the ontology of the eCatalog ontological model, the concepts associated to $toto$'s rated items are extracted and inserted in the ontological profile. For example, item 2 (Star War Episode II) is instance of concept 4; which implies that concept 4 gets added to the ontological profile. As the preferences in ontology filtering are elicited using a 5 stars rating (Section 2.5.1), the maximum possible rating is 5. Thus,

the ratings become normalized by dividing all of the user's ratings by 5. Following this, and because item 8 (Marry Poppins) is also instanced this concept, the score of concept 4 becomes as $((5/5)+(1/5))/2= 0.6$ (Equation 3.1).

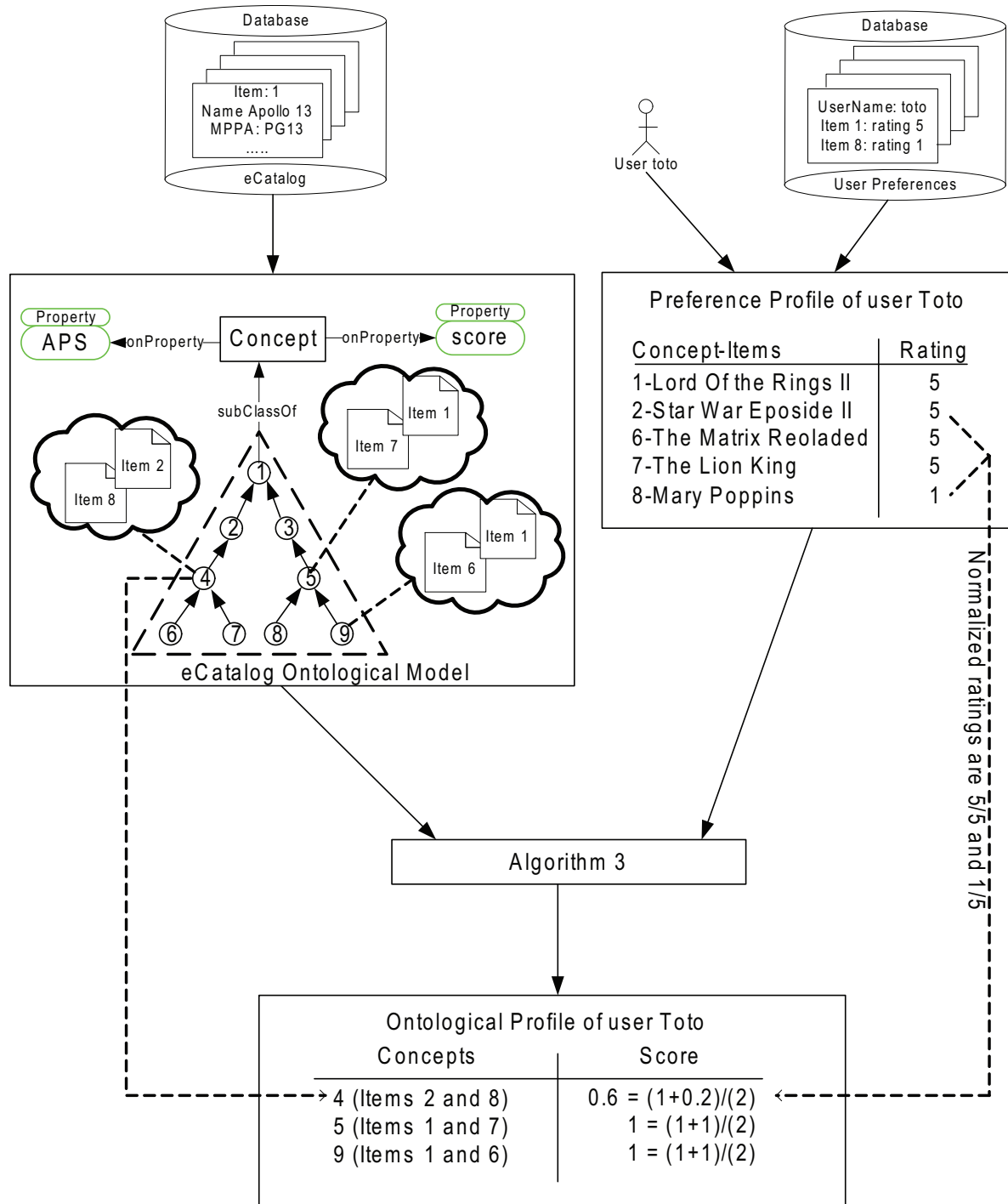


Figure 5.1: Example of a user ontological profile for user toto.

5.3 Inference from one concept to another

In an eCommerce environment, it is very unlikely that a user is able (or willing) to express her preferences on enough items in order for the user's ontological profile to contain all the concepts of the ontology. As a consequence, this leads to an incomplete preference model that seriously limits the predictions made by the recommender system. Given an ontology, this section shows that it is possible to infer missing preferences by transferring the score from one concept to another, as long as a chain connecting both concepts exists. Thus, when a user's scores for certain concepts are known more precisely, ontology filtering can derive a personalized score for the other concepts by propagation. In this dissertation, the notation $S(y|x)$ is used to denote the inferred score of concept y knowing only the score of x .

For example, imagine a situation with two concepts x and y , but where only $S(x)$ is known. To propagate the score from concept x to y , a link between these two concepts must be found. In a tree structure, there are three cases to consider: when y is a parent of x ($x \subset y$, Fig. 5.2a), when x is a parent of y ($y \subset x$, Fig. 5.2b), or when x is neither a parent nor a child of y ($x \not\subset y \wedge y \not\subset x$, Fig. 5.2c).

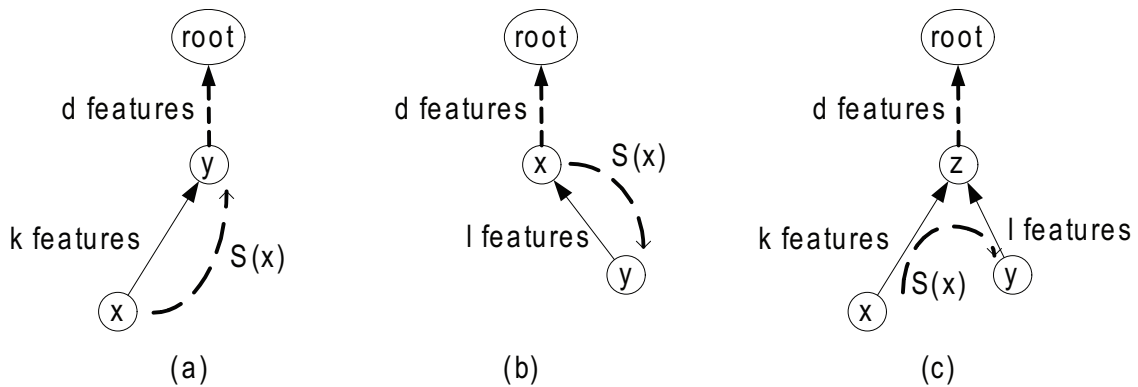


Figure 5.2: Possible chains in a tree structure between the concepts x and y : (a) x is a child of y , (b) x is a parent of y , and x is neither a parent nor a child of y (c).

Thus, the first task in the propagation is to identify the chain $C(x, y)$ that contains both concepts. To minimize the amount of propagation, we construct the chain through the lowest common ancestor. Informally, in a tree structured graph, the lowest common ancestor of nodes x and y , $LCA(x, y)$, is defined as the node farthest from the root that is an ancestor to both u and v [Bender et al., 2005].

In an eCommerce environment, the structure of the ontology is often a tree, but directed acyclic graphs can also occur. As concepts in a DAG can have multiple parents, there can exist more than one chain between pairs of concepts. As a consequence, there can be several LCA nodes; in fact, the number of LCA nodes can grow exponentially with the size of the graph. Fortunately, this number tends to be small in reality, as most concepts have only a few parents. For example, a concept in the WordNet ontology has on average 1.03 parents.

5.3.1 Finding the lowest common ancestor

Finding the lowest common ancestor is also a fundamental problem in graph theory, and has been widely looked at over the years (a good survey can be found in [Bender et al., 2005]). Formally, in a tree structured graph, the lowest common ancestor between nodes x and y , $LCA(x, y)$, is defined as the deepest node that is an ancestor to both x and y , where the depth of a node x is the length of the longest path from the root to x . The structure of a tree guarantees the existence and the uniqueness of a lowest common ancestor between any pair of concepts. However in DAG, there can be many lowest common ancestors, as they are potentially many paths from one concept to the root. In a DAG, the lowest common ancestors of nodes x and y , $LCAs(x, y)$, are defined as the deepest nodes that are ancestors to both x and y , and obtained from distinct chains linking x and y . Figure 5.3 illustrates an ontology with a DAG structure. Take for example concepts h and i . There are two chains linking these concepts: $\langle h, d, b, e, i \rangle$ and $\langle h, e, i \rangle$. Thus, the lowest common ancestors to both concepts h and i are concepts b and e , where b is the LCA of the former chain and e to the latter.

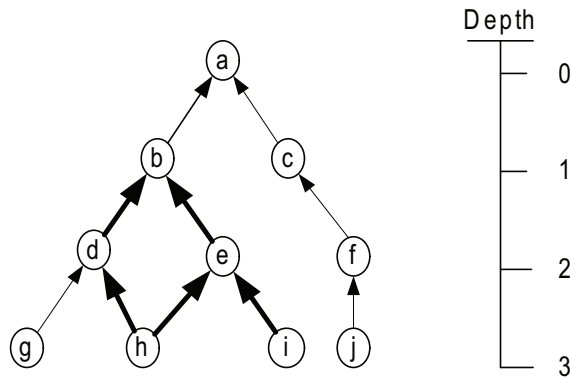


Figure 5.3: Example of paths for concepts h and i to the root concept a , where the path are in bold lines.

Finding the lowest common ancestor in a tree structured graph is not a hard problem. [Bender et al., 2005] proposed an algorithm based on the *Euler tour* and the *range minimum query*, RMG, that can solve the problem with linear space requirement and $\mathcal{O}(1)$ query time. The algorithm is as follows. First, all n nodes of the tree are visited using an Euler tour in order to construct the array A of size $m = 2n - 1$. A is then virtually split in k buckets of size $\log(m)/2$, and an array A' is constructed that contains the minimum value of each bucket. The idea behind the bucket strategy is to split the big array A in small intervals in order to reduce the search space. Third, a lookup table T_A is constructed that will contain the RMG for each bucket separately. At the same time, another lookup table $T_{A'}$ is built that will answer the RMG for the array A' . Using this construction, the lowest common ancestor is answered by three look queries, two in the table T_A and one in table $T_{A'}$.

Unfortunately, finding the lowest common ancestors in DAG is much harder. It requires generating the ancestor-existence matrix, which requires at least transitive closure time [Bender et al., 2005]. Using this ancestor-existence matrix, all the LCAs can be found with

a complexity $\mathcal{O}^{2.688}$. However, it remains theoretical, and its explanation goes far beyond the content of this dissertation.

This dissertation proposes a much simpler algorithm that is easily implementable, and has successfully been implemented for this thesis. The first idea of the algorithm is to take advantage of the fact that the lowest common ancestor is the common ancestor with the biggest depth. Second, as a DAG can have several LCA, a heuristic is proposed for selecting which one to use during the propagation of the score. The heuristic for selecting the lowest common ancestors n of concepts x and y uses the following two functions:

- the reinforcement $r(n)$, given as the number of different paths leading to n leading from x and y , and
- the depth $depth(n)$, given as the length of the longest path between the root and n .

For each possible LCA node n , a heuristic value $r(n) * 2^{depth(n)}$ is computed, and the node with the highest value is chosen as the best lowest common ancestor. This heuristic is based on the idea that while the amount of propagation should be limited, if a node appears in many different connections between concepts x and y , then it can become more meaningful as a connection.

Algorithm 4: *Finding the lowest common ancestor in a directed acyclic graph for a user u .*

Inputs: An ontology λ ; and two concept x and y

Outputs: The lowest common ancestor lca for concepts x and y

```

1  $LCAs \leftarrow \{\phi\}$ 
2  $Chains_x \leftarrow$  generate all the chains from concept  $x$  to the root
3  $Chains_y \leftarrow$  generate all the chains from concept  $y$  to the root
4 forall  $chain_x \in Chains_x$  do
5   forall  $chain_y \in Chains_y$  do
6      $ancestors \leftarrow \{chain_x\} \cap \{chain_y\}$ 
7      $lca \leftarrow z | arg_{z \in ancestors} \max(depth(z))$ 
8      $LCAs \leftarrow LCAs \cup lca$ 
   end
end
9 return  $lca | arg_{lca \in LCAs} \max(r(lca) * 2^{depth(lca)})$ 

```

Formally, Algorithm 4 describes in details the algorithm used for finding which lowest common ancestor to use in a DAG. Step 1 creates and initializes a set $LCAs$ that will contains all the possible lowest common ancestors of both nodes x and y . For each of these nodes, step 2 and 3 respectively find all the chains from these concepts to the root of the graph. For example, step 2 generates the set $\{\langle h, d, b, a \rangle, \langle h, e, b, a \rangle\}$ for concept h , and $\{\langle i, e, b, a \rangle\}$ for concept i of Figure 5.3. In step 6, the intersection between pairs of chains of each concept is computed in order to find the common ancestors to both concept x and y . Note that step 6 initially converts the chains which are sequences of nodes into sets of nodes. From these ancestors, step 7 selects the lowest common ancestor lca as the node with the biggest depth. Then, lca is added to the set of all the common ancestors, and

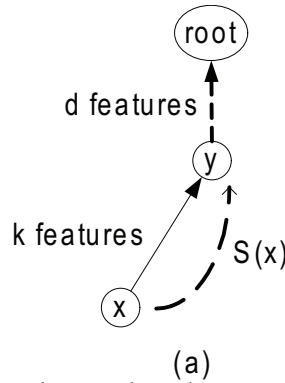
steps 6 to 8 are reiterated on all possible combinations of chains linking concepts x and y . Finally, step 9 returns the node that maximizes the heuristic function among all possible lowest common ancestors.

To continue the previous example, Algorithm 4 starts with the first chain of each node (i.e.: $\langle h, d, b, a \rangle$ and $\langle i, e, b, a \rangle$). The common ancestors set obtained from these two sequences is $\{b, a\}$, which implies that step 7 would select node b as lowest common ancestor (i.e.: depth of node $b=2 >$ depth of node $a=1$). As node h also possesses the chain $\langle h, e, b, a \rangle$, the algorithm reiterates and finds that node e is also a lowest common ancestor. Thus, the algorithm finds that the lowest common ancestors for nodes h and i are $\{b, e\}$. Finally, the heuristic function is applied these nodes, and the heuristic values $3 * 2^1 = 6$ and $2 * 2^2 = 8$ are respectively obtained from nodes b and e . As a consequence Algorithm 4 selects node e as the lowest common ancestor.

As a node can have multiple parents, the number of chains can grow exponentially with the size of the graph. This implies that the complexity of Algorithm 4 can also grow exponential with the size of the graph. Fortunately, as most concepts have only a few parents, the worst complexity hardly ever occurs in real life. Moreover, experiments on the WordNet and GeneOntology showed that Algorithm 4 can produce results in real time.

5.3.2 Upward inference

This situation arises when there is a path going from concept x to its k^{th} parent y ($x \subset_k y$, Figure 5.2a). From the graph construction, both concepts have d features in common, but the concept x has an extra k features that differentiates it from its ancestor.



By definition of the model, we know that the score of a concept depends on the features defining that concept (A1). Informally, it means that the score of y can be computed knowing the score of x , $S(y|x)$, by looking at the ratio of features they have in common and liked by the user. Formally, $S(y|x)$ is defined as follows:

$$S(y|x) = \alpha(x, y)S(x), \quad (5.1)$$

where $\alpha(x, y)$ is the coefficient of generalization that contains the ratio of features in common which are liked according to their respective distribution. Note that the assumption (A1) and (A2) of the score function implies the following property.

Property 5.1 *Let x, y be two concepts in an ontology λ , where the x is a descendant of y ($x \subset y$). Then, $S(y) = S(y|x)$.*

Proof 5.1 By definition, coefficient $\alpha(x, y)$ is the coefficient of generalization that contains the ratio of features in common to both concepts x and y , and which are liked according to their respective distribution. Assumption (A1) implies that a user likes the concept y because of the features found between the root and y . As y is on the path from the root to x , $\alpha(x, y)S(x)$ measures how much a user likes the features found on the path between the root and y . As a consequence, $S(y) = S(y|x)$ \square .

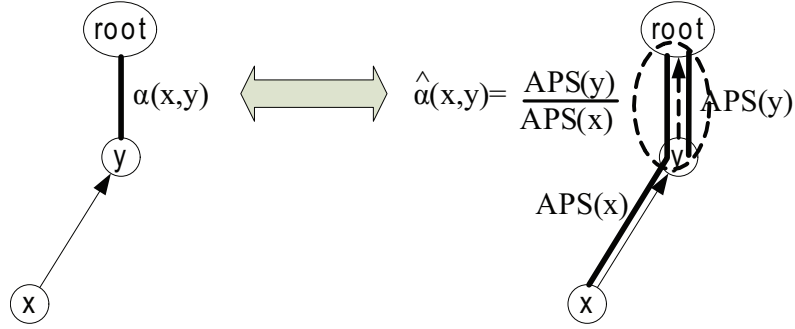


Figure 5.4: Illustration of the correspondences between $\alpha(x, y)$ and $\hat{\alpha}(x, y)$.

Obviously, α is unknown in practice. Furthermore, it is unfeasible to elicit it from the user, as some features are implicit in the edges of the ontology. Instead, this dissertation proposes to estimate α by using the a-priori score captured by the concepts in the ontology (Figure 5.4). By definition, the $APS(x)$ measures the a-priori score that an average user will give to concept x . As the score of a concept is constructed from the features making it, $APS(x)$ reflects how much an average user likes the features making x . Thus, the coefficient of generalization can be estimated as the ratio of a-priori scores:

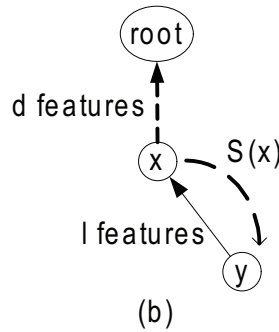
$$\hat{\alpha}(x, y) = \frac{APS(y)}{APS(x)}. \quad (5.2)$$

As a consequence, the upwards inference from a concept x to a concept y can be estimated as follows:

$$\hat{S}(y|x) = \hat{\alpha}(x, y)S(x). \quad (5.3)$$

5.3.3 Downward inference

Inversely, we have the case when y is the l^{th} descendant of x ($y \subset x$).



From the previous result, it is very tempting to assume that $S(y|x) = \beta S(x)$, where β is a coefficient of specialization that contains the ratio of features in common. However, this reasoning is not compatible with the third assumptions - (A3) user are risk averse. To understand this assumption, imagine that the score of the object is equal to the maximum price a user is willing to pay. Consider two concepts x and y , where y has one more feature than x . Now consider two users A and B such that A values x more than B does. This does not automatically mean that A will also attach a higher value to the extra feature that distinguishes y from x . Notice also that when we were traveling upwards, we were considering super concepts, which means we were removing known features whose contributions to the score are likely to be proportional to 1. However, when traveling downwards, we are adding new (unknown) features to the concept. Therefore, the score of each new feature need to be considered separately from known one. Using the second assumption that states that features contribute independently to the score, $S(y|x)$ becomes as follows:

$$S(y|x) = S(x) + \beta(y, x), \quad (5.4)$$

where $\beta(y, x)$ is the coefficient of specialization that contains the score of the features contained in concept y but not in x (Figure 5.5). Again, β can be estimated using the a-priori score:

$$\hat{\beta}(y, x) = APS(y) - APS(x). \quad (5.5)$$

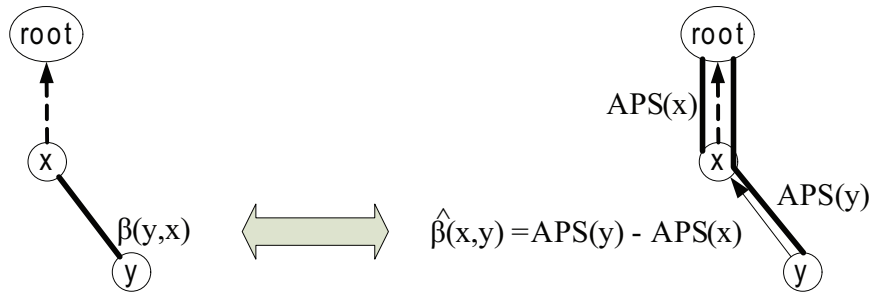


Figure 5.5: Illustration of the correspondences between $\beta(y, x)$ and $\hat{\beta}(y, x)$.

5.3.4 Upward & downward inference

Finally, this section considers the case when there is no direct path between concepts x and y .

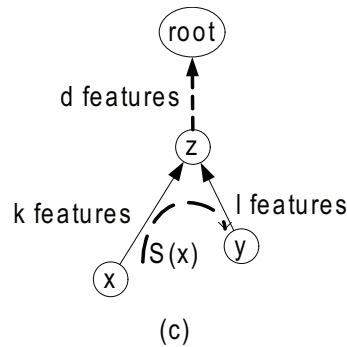


Figure 5.2(c) shows that in order to transfer the preference, the score needs to be carried up to the lowest common ancestor $z = LCA_{x,y}$, and then down to the concept y . Equation 5.4 implies that $S(y|z) = S(z) + \beta(y, z)$, while Equation 5.1 defines $S(z|x)$ as equal to $\alpha(x, z)S(x)$. Furthermore, as Property 5.1 implies that $S(z) = S(z|x)$, the inferred score of concept y knowing z becomes as follows:

$$S(y|z) = S(z) + \beta(y, z) = S(z|x) + \beta(y, z) = S(y|x). \quad (5.6)$$

Furthermore, Equations (5.1), (5.4), and (5.6) imply that the score of any concept y can be inferred from the score of concept x as follows.

$$\begin{aligned} S(y|x) &= S(z|x) + \beta(y, z) \\ &= \alpha(x, z)S(x) + \beta(y, z) \\ &= \alpha(x, LCA_{x,y})S(x) + \beta(y, LCA_{x,y}) \end{aligned} \quad (5.7)$$

Unfortunately, this equation is useless in practice as the coefficients α and β are unknown and cannot be elicited. Using Equations (5.2) and (5.5), the inferred score of concept y knowing the score of x can be estimated as follows:

$$\hat{S}(y|x) = \hat{\alpha}(x, LCA_{x,y})S(x) + \hat{\beta}(y, LCA_{x,y}) \quad (5.8)$$

where $LCA_{x,y}$ is the lowest common ancestor of both concepts x and y , and obtained using Algorithm 4.

5.4 Finding the closest concept

The inference function defined in Equation 5.7 transfers the score from one concept to another through their lowest common ancestor. In practice, the user's ontological profile usually contains more than one concept. This raises the following question; from which concept should the inference start?

By construction, the inferred score is made from the user's score and the a-priori scores. As the a-priori scores are estimations of how much an average user likes a given concept, the inference mechanism should minimize the use of it. Thus, it needs to find the closest concepts to the unknown one so that it can minimize the propagation error. Finding the closest concept in an ontology is not as easy as it may seem, and it requires evaluating the similarity between pairs of concepts.

Evaluating semantic similarity between concepts is a fundamental task in the ontology community. Most authors have focused their research on hierarchical ontologies (HO), which is not surprising as most ontologies are made of *is-a* relationships ($\simeq 82\%$ of all the relations in WordNet 2.0 are *is-a* relationships, while GeneOntology has $\simeq 87\%$). Furthermore, [Maguitman, 2005] has shown that a similarity metric defined on a hierarchical ontology can be generalized to any kind of ontology by using a weighted combination of the *is-a* metrics. Thus, the problem of evaluating the semantic similarity between concepts in any kind of ontology can be simplified to hierarchical ontologies. To this day, there exist two main approaches for estimating similarity between concepts in a hierarchical ontology:

the edge based approach and the node based approach. Unfortunately, existing approaches fail to achieve high correlation with human ratings, while experiments on the GeneOntology have shown that existing techniques are sensitive to the topology of the ontology.

To improve the computation of pairwise similarities, this thesis defines a novel similarity measure for ontologies called *Ontology Structure based Similarity* (OSS). As shown in Figure 5.6, OSS computes the similarity between two concepts x and y in four steps. First, the *score* of the concept y is inferred from x using Equation 5.7. From this inferred score, OSS computes how much of the score has been transferred between these two concepts, $T(x, y)$, and then takes the lower bound value $\lfloor T(x, y) \rfloor$. Finally, this lower bound value is transformed into a distance value $D(x, y)$ using the logarithm function.

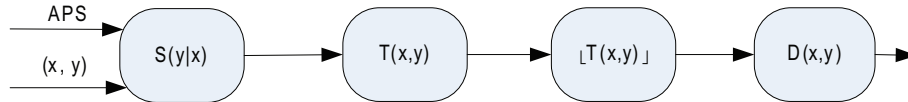


Figure 5.6: Illustration of the OSS approach.

5.4.1 Existing measures of similarity

Many techniques have been proposed for evaluating the semantic similarity between two concepts in an ontology. At the same time, authors have looked at this problem from either a *distance* or *similarity* point of view. These approaches are duals, as the *similarity* can be defined as $1 - \text{distance}$, when distance values are normalized to $[0..1]$. Concretely, existing techniques can be categorized as follows:

The edge based approach is the traditional, most intuitive, and simplest similarity measure. It computes the distance between two concepts based on the number of edges found on the path between them. [Resnik, 1995] introduced a variant of the edge-counting method, converting it from a distance to a similarity metric by subtracting the path length from the maximum possible path length:

$$\text{sim}_{EDGE}(x, y) = (2 \times D) - \text{len}(x, y), \quad (5.9)$$

where a and b are concepts in the ontology, D is the maximum depth of the HO, and $\text{len}(x, y)$ is the shortest path between concepts a and b . Another popular variant of the edge based approach is the metric proposed by [Leacock, 1997], which scales the shortest path by twice the maximum depth of the HO.

$$\text{sim}_{LEACOCK}(x, y) = -\log \left(\frac{\text{len}(x, y)}{2 \times D} \right) \quad (5.10)$$

The node based approach was proposed by [Resnik, 1995] to overcome the drawbacks of the edge-counting approach, which considers the distance uniform on all edges. Resnik defined the similarity between two concepts as the *information content* of the lowest common ancestors to both concepts. The information content of a concept c , $\text{IC}(c)$, is defined as the negative log likelihood of the probability of encountering

an instance of c , i.e. $IC(c) = -\log P(c)$. The intuition behind the use of the negative likelihood is that the more probable a concept is of appearing, then the less information it conveys. Formally, the similarity is defined as follows.

$$sim_{RESNIK}(a, b) = \max_{z \in LCA(x, y)} IC(z) \quad (5.11)$$

While Resnik defined the similarity based on the shared information, [Lin, 1998] defined the similarity between two concepts as the ratio between the amount of information needed to state the commonality between these two concepts and the information needed to fully describe them.

$$sim_{LIN}(x, y) = \frac{2 \times \max_{z \in LCA(x, y)} IC(z)}{IC(x) + IC(y)} \quad (5.12)$$

Hybrid approaches combine both approaches defined above. [Jiang and Conrath, 1997] proposed a combined model that is derived from the edge based notion by adding the information content as a decision factor. They defined the link strength between two concepts as the difference of information content between them. Following this, Jiang's distance metric is defined as follows:

$$dist_{JIANG}(x, y) = IC(x) + IC(y) - 2 \times \left(\max_{z \in LCA(x, y)} IC(z) \right). \quad (5.13)$$

5.4.2 Transferring the score from one concept to another

The intuition is that the *distance* between two concepts x and y is correlated to the amount of score being transferred between them. Informally, the more score that can be transferred from one concept to another, then the more similar these concepts are. Moreover, OSS takes into account the fact that users are pessimistic by considering the fact that unknown features should not contribute to the transfer.

Formally, a distance measure is a real valued function that we would like to satisfy the following axioms:

- *identity*: $D(x, y) = 0 \Leftrightarrow a = b$
- *normalization*: $0 \leq D(x, y) \leq 1$
- *triangle inequality*: $D(x, z) \leq D(x, y) + D(y, z)$

It is otherwise customary to also include a symmetry axiom; however, one innovation of this work is that distance can be asymmetric. Thus, this dissertation does not include this axiom.

Furthermore, the distance between two concepts must be independent of any particular amount of score that is being assigned to the concept. In particular, it must be applicable to any score in the admissible range $[0..1]$. Thus, distance can be related to the transfer of score only in a multiplicative, but not in an additive way. Following this, we define the

transfer of score from a concept x to y , $T(x, y)$, is defined as the amount of score being propagated between x and y :

$$S(y|x) = S(x) \times T(x, y) \Rightarrow T(x, y) = \frac{S(y|x)}{S(x)}. \quad (5.14)$$

Using Equations 5.1, 5.4, $T(x, y)$ can be decomposed as follows.

$$\begin{aligned} T(x, z) &= \alpha(x, z) & x \subset z & \nearrow \\ T(z, y) &= 1 + \frac{\beta(z, y)}{S(z)} & z \subset y & \searrow \end{aligned} \quad (5.15)$$

In an eCommerce environment, $S(z)$ is usually unknown, which renders the computation of the downwards transfer impossible. However, under the assumption that the score of any concept is uniformly distributed between 0 and 1 (Section 3.3.2), the expected score of a concept is in fact equal to $1/2$. Thus, the downwards transfer $T(z, y)$ can be approximated as $1 + 2\beta(z, y)$.

As the transfer of score $T(x, y)$ is multiplicative, this implies the following transitive property.

Property 5.2 *Let x, y, z be three concepts in an ontology λ , where the z is the lowest common ancestor to both concepts x and y (i.e. $y \subset z$ and $x \subset z$). Then, $T(x, y) = T(x, z) \times T(z, y)$.*

Proof 5.2 *From Equation 5.14, $T(x, y)$ is defined as $S(y|x)/S(x)$. Using Equation 5.15, $T(x, y)$ becomes as follows.*

$$T(x, y) = T(x, z) \frac{S(y|x)}{S(z|x)} \quad (5.16)$$

As concept x is a descendant of y , Property 5.1 simplifies the above equation to:

$$T(x, y) = T(x, z) \frac{S(y|z)}{S(z)} \quad (5.17)$$

Finally, as Equation 5.14 implies that $S(y|z)/S(z) = T(z, y)$, $T(x, y)$ simplifies to $T(x, z)T(z, y)$. \square

5.4.3 Getting a lower bound value

When judging the distance between two concepts x and y , the transfer of score from x to y should not attempt to predict the expected value of the score assigned to y , but a lower bound on it in order to model the fact that unknown features do not contribute. For upward propagation, the factor α derived from the a-priori scores reflects this correctly, as the APS of an ancestor is constructed as a lower bound on scores of its descendants.

On the other hand, for downward propagation, the term β derived from the a-priori score reflects that translation from the lower bound to the expected value at the descendants. Thus, in order to produce a lower bound, the relation has to be inverted.

Consequently, a lower bound on the transfer of score between concepts x and y , $[T(x, y)]$, can be defined as follows:

$$\begin{aligned} [T(x, z)] &= \alpha(x, z) & x \subset z & \nearrow \\ [T(z, y)] &= \frac{1}{1+2\beta(z, y)} & z \subset y & \searrow, \end{aligned} \quad (5.18)$$

where concept z is the lowest common ancestor to both concepts x and y .

5.4.4 The ontology structure based similarity - OSS

The *ontology structure based similarity* between two concepts x and y is defined as $1 - D(x, y)$, where $D(x, y)$ is a distance measure between the concepts. To guarantee that $D(x, y) \geq 0$, and in order to satisfy the identity relation, $[T(x, y)]$ is scaled into a distance by taking its negative logarithm: $D(x, y) = -\log[T(x, y)]$. However, this distance would not be normalized to fall in the interval $[0..1]$. By using $\max D$ as longest distance between any two concepts in the ontology, the normalized distance becomes as follows.

$$D(x, y) = -\frac{\log([T(x, y)])}{\max D} \quad (5.19)$$

Thus, the distance measure satisfies the normalization. Such a logarithmic measure has also been used elsewhere for distance and similarity measure [Jiang and Conrath, 1997] [Lin, 1998] [Resnik, 1995]. To ease the reading of this dissertation, the proof that Equation 5.19 satisfies the transitive property has been added to Appendix B.

Finally, using the transitive property 5.2, the distance between any concepts x and y in an ontology is formally defined as follows:

$$D(x, y) = \frac{\log(1 + 2\beta(y, z)) - \log(\alpha(x, z))}{\max D}, \quad (5.20)$$

where z is the lowest common ancestors to both concepts x and y . As mentioned previously, the coefficients α and β cannot be excited in practice. Fortunately, using Equations (5.18) and (5.19), the distance between any two concepts in the ontology x and y can be estimated as follows.

$$\hat{D}(x, y) = \frac{\log(1 + 2\hat{\beta}(y, LCA(x, y))) - \log(\hat{\alpha}(x, LCA(x, y)))}{\max D}, \quad (5.21)$$

where $LCA(x, y)$ is the lowest common ancestor to both concepts x and y and is computed using Algorithm 4.

5.4.5 Example

Table 5.1 illustrates the distance computation between the concepts x and z of the ontology found in Figure 5.7.

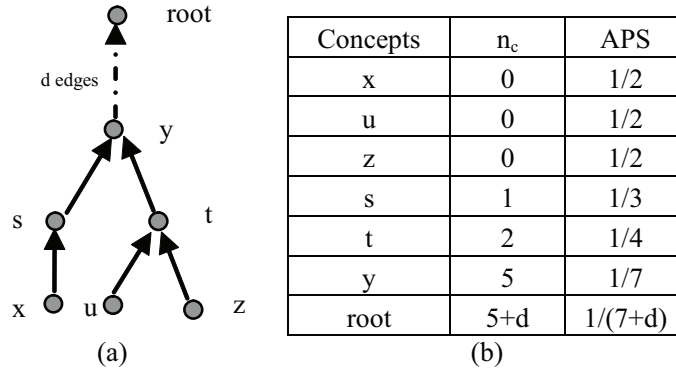


Figure 5.7: (a) a simple ontology λ and its APSs (b).

| Concepts | Direction | Transfer | Distance | $D(x, y)$ |
|------------------------|----------------|--|----------------------------------|------------------------------|
| $x \rightsquigarrow z$ | $x \nearrow y$ | $\hat{\alpha} = \frac{1}{7} / \frac{1}{2}$ | $-\log\left(\frac{2}{7}\right)$ | $\simeq \frac{2.58}{\max D}$ |
| | $y \searrow z$ | $1 + 2\hat{\beta} = \frac{24}{14}$ | $\log\left(\frac{24}{14}\right)$ | |

Table 5.1: Distance between concepts x and z from the ontology found in Figure 5.7, where the distance is computed using Equation 5.21.

5.5 The overall inference process

Equation (5.7) showed that it is possible to infer the score of any concept y from a concept x with a known score. However, in an eCommerce environment, a user typically rates more than one item. This implies that more than one concept will have a score assigned to it, and a choice must be made to discriminate these concepts. Thus, Equation 5.21 is used to find the closest concept x from y so that the inference has the smallest error.

Algorithm 5 now explains in details how the scores of all the concepts of the eCatalog ontological model are inferred. First, the user's ontological profile UOP is created from the user's preference profile UPP using Algorithm 3. Step 2 simply checks that the UOP contains at least one score. If it does not, then the algorithm terminates at step 3. Otherwise, step 4 extracts all the concepts from the ontology without score, and inserts them in the new set UOP' . For each concept y in UOP' , step 7 returns the set of all the closest concepts with a score to a concept y using Equation 5.21. As it is assumed that users are risk averse, step 8 finds the closest concept x that will infer the minimum score. Using Equation 5.7, step 9 then sets the score of y with the inferred value transferred from the score of x . Note that steps 7 and 8 are not necessary when the user's ontological profile contains only one concept. In this situation, step 11 directly infers the score of concept y from the value of the only concept that contains preferences. Once all the scores have been computed, the complete user's ontological profile (which is the union of the sets UOP and UOP') is returned.

Algorithm 5: *Building a complete user's ontological profile for a user u .*

Inputs: The user's preference profile UPP , and the eCatalog ontological model $eCOM$.

Outputs: A complete user's ontological profile associated to the eCatalog ontological model $eCOM$.

Functions: $\text{buildOntologicalProfile}(UPP, eCOM)$ implements Algorithm 3.

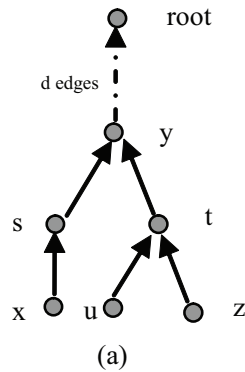
```

1  $UOP \leftarrow \text{buildOntologicalProfile}(UPP, eCOM)$ 
2 if  $|UOP| = 0$  then
3   | STOP-NO-PREFERENCES-FOR-USER
   end
4  $UOP' \leftarrow eCOM / \{c | c \in UOP\}$ 
5 forall  $y \in UOP'$  do
6   | if  $|UOP| > 1$  then
7     |  $X \leftarrow \{z | \arg_{z \in UOP} \min(\hat{D}(z, y))\}$ 
8     |  $x \leftarrow x | \arg_{x \in X} \min(S(y|x))$ 
9     |  $S(y) \leftarrow S(y|x)$ 
   | end
10  | else
11  |  $S(y) \leftarrow S(y | (x | x \in UOP))$ 
   | end
   end
12 return  $UOP \cup UOP'$ 

```

5.5.1 Example

To illustrate Algorithm 5, re-consider the simple ontology λ shown below. This ontology, along with the APSs, will be used as the eCatalog ontological model.



| Concepts | n_c | APS |
|----------|-------|---------|
| x | 0 | 1/2 |
| u | 0 | 1/2 |
| z | 0 | 1/2 |
| s | 1 | 1/3 |
| t | 2 | 1/4 |
| y | 5 | 1/7 |
| root | 5+d | 1/(7+d) |

(b)

Imagine a user *toto* with a unique preference on the movie Appolo 13. Further imagine that *toto* assigns the maximum rating of 5 to this movie, which happens to be an instance of concept x . Using the ontology shown above, Algorithm 3 generates a user ontological profile that contains the single tuple $S(x) = 1$. Finally, Algorithm 5 builds the complete user's ontological profile contained in Table 5.2.

| Set | Concept | Propagation | Score |
|--------|---------|---------------------|--|
| UOP | x | - | 1 |
| UOP' | s | \nearrow | $1 \times \left(\frac{1/3}{1/2}\right) = \frac{2}{3}$ |
| | z | \nearrow | $1 \times \left(\frac{1/7}{1/2}\right) = \frac{2}{7}$ |
| | root | \nearrow | $1 \times \left(\frac{1/(7+d)}{1/2}\right) = \frac{2}{(7+d)}$ |
| | t | $\nearrow \searrow$ | $1 \times \left(\frac{1/7}{1/2}\right) + \left(\frac{1}{4} - \frac{1}{7}\right) = \frac{11}{28}$ |
| | u | $\nearrow \searrow$ | $1 \times \left(\frac{1/7}{1/2}\right) + \left(\frac{1}{2} - \frac{1}{7}\right) = \frac{9}{14}$ |
| | y | $\nearrow \searrow$ | $1 \times \left(\frac{1/7}{1/2}\right) + \left(\frac{1}{2} - \frac{1}{7}\right) = \frac{9}{14}$ |

Table 5.2: Toto's Ontological Profile constructed using Algorithm 5.

5.6 Recommending items to a user

Once the score of all the concepts have been inferred, recommending items becomes straightforward. The idea is to recommend the best N items that the user has not seen, and which have the highest probability of being liked. This is commonly called the top- N recommendation strategy.

Moreover, the majority of users are known to like popular items. Non-personalized recommender systems such as Digg.com and MovieFinner.com rely on this fact to recommend items to their users. This is called the aggregated ratings strategy (see Section 2.2.1). Take for example a user who wants to go to the cinema, but who doesn't know what to watch. Further imagine that among the alternatives there are the new Harry Potter - The Order of the Phoenix and the French movie - Fragile. In this situation, a recommender system recommending the Harry Potter movie will be right in most cases, as this movie is number 1 in the box office. Note that collaborative filtering also uses the average rating of an item when predicting the rating of that item to a user (Equation 2.25).

Following these observations, ontology filtering will recommend perviously unseen items that are instance of the concepts with the highest score, and which are also popular. As a consequence, this thesis defines the *hybrid score* of a concept c , $HS(c)$ as follows:

$$HS(c) = \rho S(c) + (1 - \rho)P(c), \quad (5.22)$$

where ρ is a personalization coefficient that ranges in the interval $[0, 1]$, and $P(c)$ is the popularity of a concept also in the interval $[0, 1]$. The popularity of concept c is simply computed as the average of the normalized ratings of all the users on the items which are instanced of concept c .

$$P(c) = \frac{\sum_{i \in c, u \in U} \hat{R}_{u,i}}{n_{ratings}}, \quad (5.23)$$

where U is the set of all the users in the database, $\hat{R}_{u,i}$ is the normalized rating assigned by user u on item i , and $n_{ratings}$ is the total number of ratings for concept c . Note that if the ρ coefficient is set to 0, then the popularity score is equivalent to the non-personalized recommender systems but not similar, as concepts can contain more than one concept. Obviously, the personalization coefficient ρ can be learnt on a training set, but this dissertation

sets the coefficient to $1/2$ for two reasons. First, and for practical reasons in eCommerce environment, training is usually not feasible as the ratings on the items change too frequently. Second, it allows to introduce a good mix of diversity in the recommendations, while maintaining the personalized aspect.

Algorithm 6: *Recommending the top-N items to user u.*

Inputs: The complete user's ontological profile UOP , the number of items N wanted in the recommendation set, and a list of possible alternatives, $Items$.

Outputs: The top-N recommendations.

Functions: $sort(ratings)$ sorts a list $ratings$ (composed of tuple $\langle item, hybridscore \rangle$) on decreasing value of the hybrid score.

```

1 ratings ← []
2 topN ← []
3 counter ← N
4 forall i ∈ Items do
5   | c ← c | c ∈ UOP ∧ i ∈ c
6   | ratings ← ⟨i, HS(c)⟩
   end
7 ratings ← sort(ratings)
8 while counter > 0 do
9   | topN ← topN ∪ ratings[0]
10  | ratings ← ratings/ratings[0]
11  | counter--
   end
12 return topN

```

Formally, ontology filtering recommends the top-N items to a user using Algorithm 6. For each item i to be recommended from $Items$, step 5 extracts the concept from the user's ontological profile UOP from which the item is instance of, while step 6 computes the hybrid score of this concept using Equation 5.22. Once all the hybrid scores have been computed, step 7 sorts the items on decreasing values of the hybrid score computed previously. Then, steps 8 to 11 select the best N items, and insert them in the $topN$ list. The algorithm terminates by returning the Top-N recommendation list $topN$.

5.6.1 Example

Re-consider the previous example illustrated by Figure 5.8. Let's imagine three items i, j, k respectively instanced of concepts $t, u,$ and y . Further imagine that these three concepts have the following popularity score: $P(t) = \frac{3}{6}$, $P(u) = \frac{4}{6}$, and $P(y) = \frac{5}{6}$. Note that in practice, these popularity scores are computed off-line using Equation 5.23.

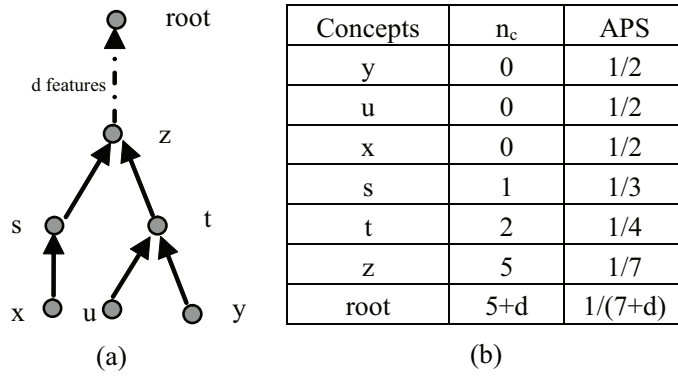


Figure 5.8: (a) a simple ontology λ and its corresponding a-priori score (b).

Using the scores in Table 5.2, step 6 of Algorithm 6 finds that the hybrid score of concepts t , u and v is $\frac{25}{56}$, $\frac{55}{84}$, and $\frac{31}{42}$. Table 5.3 displays the order of the items in the $top - N$ list when step 7 is completed.

| Position in <i>ratings</i> | Item | Concept | Score | Popularity | Hybrid Score |
|----------------------------|------|---------|-----------------|---------------|---|
| 0 | k | y | $\frac{9}{14}$ | $\frac{5}{6}$ | $\frac{1}{2} \frac{9}{14} + \frac{1}{2} \frac{5}{6} = \frac{31}{42} \simeq 0.74$ |
| 1 | j | u | $\frac{9}{14}$ | $\frac{4}{6}$ | $\frac{1}{2} \frac{9}{14} + \frac{1}{2} \frac{4}{6} = \frac{55}{84} \simeq 0.65$ |
| 2 | i | t | $\frac{11}{28}$ | $\frac{3}{6}$ | $\frac{1}{2} \frac{11}{28} + \frac{1}{2} \frac{3}{6} = \frac{25}{56} \simeq 0.45$ |

Table 5.3: Items with their position in the list *ratings* when Algorithm 6 is at step 8.

Thus, if the recommender system applies the top-1 recommendation policy, then it is item k that would be recommended, as it is instance of the concept with the highest hybrid score (i.e: $HS(y) \simeq 0.74$). Similarly, if the top-2 recommendation policy was used, then it would have been items k and j that were recommended; and so forth.

Chapter 6

Learning the structures of the ontologies

The ontologies are fundamental elements of this dissertation. They are used to create the eCatalog ontological model, construct the users' ontological profiles, and infer missing preferences. However, previous chapters assumed the existence of these ontologies, without considering their construction. With eCatalogs continuously changing, new techniques are required to build these ontologies in real time, and without any expert intervention.

This chapters focuses on this problem, and shows that it is possible to learn these ontologies autonomously by using hierarchical clustering algorithms. Rather than using the same ontology for all users, another algorithm is defined that personalizes the ontology based on the user's preferences. This personalization lets the system find the ontology that best matches the user's preferences, which increases the recommendation accuracy. Finally, it is suggested that recommendation accuracy can be further improved by allowing multiple inheritance edges in the clustering tree. Thus, a third algorithm is presented that can construct ontologies with multiple hierarchical structure.

6.1 Introduction

In some simple domains, it is feasible to create an ontology modeling the items by hand. In this situation, the features will usually be explicit, and the ontology will be created by a group of experts. Unfortunately in most cases, it is infeasible to construct such an ontology, as the content and the structure of the catalog changes too often. To overcome this problem, we propose to use clustering algorithms to generate the hierarchical inheritance structure from a set of users' ratings. It is these hierarchical structures, along with the a-priori scores, that will be used as ontologies in the eCatalog ontological models.

In the collaborative filtering research community, it is an established fact that users can be categorized in different communities, and that different communities of users behave differently. Following these observations, it is believed that using one (learnt) ontology for all users is not optimal, and that it is better to select which ontology to use based on the user's preferences. Following this, the dissertation will use several hierarchical clustering algorithms to generate a whole set of taxonomies. Then, Algorithm 1 is used to transform

these taxonomies into ontologies. Finally, ontology filtering selects the best ontology based on the user's preference profile in order to increase the recommendation accuracy.

Unfortunately, hierarchical clustering algorithms produce taxonomies with a simple tree structure. However, both the eCatalog ontological model and the inference mechanism can reason over more complex DAG structured ontologies. By generalizing hierarchical clustering algorithms, and allowing them to consider sub-optimal clusters to merge/split, it is possible to construct DAGs. This chapter introduces an algorithm that uses this idea, and experimental results in Chapter 8 show that these ontologies do improve the quality of the ontology, but at a cost of the computational complexity.

6.2 Clustering algorithms

Over the years, researchers in the data mining community have proposed many clustering algorithms in order to perform unsupervised learning. These algorithms can be classified into six categories [Lin and Chen, 2005]: fuzzy clustering, nearest-neighbor clustering, hierarchical clustering, artificial neural networks for clustering, statistical clustering algorithms, and density-based clustering. This dissertation is interested in building hierarchical ontologies. Thus, this chapter will focus on hierarchical clustering algorithms, which can build such structures.

Hierarchical clustering algorithms can in fact be categorized into two subcategories: *distance-based clustering* and *concept-based clustering*. Both approaches construct hierarchical trees, but they use very different data representation. With distance-based clustering, objects are represented in a well define space, like a vector in a 2D cartesian space. Thus, two or more objects will be assigned to the same cluster if they are close according to a given distance function. However, with concept-based clustering, objects are described by a set of concepts, where a concept is usually defined as an attribute-value pair. Given this representation, two or more objects belong to the same cluster if they share common concepts.

Distance-based clustering

When considering distance-based clustering, there are in fact two distinct ways to build a hierarchical tree: the bottom-up or top-down approach. These approaches are respectively known as the *agglomerative clustering* and the *partitional clustering*.

Partitional clustering algorithms start by assigning all the items to be clustered into a unique cluster. Then, one cluster C_j is chosen and is then further bisected into C_i , where $i \in [1, 2]$ clusters. For each item in C_j , it is assigned to the cluster C_i that optimizes a distance function. This process continues until either all the items are found on the leaf of the tree, or if the number of clusters has met a given threshold θ .

Inversely, agglomerative clusterings assign each item to its own cluster C_i , where $i \in [1, n]$. Then, the two *closest* clusters are merged into a unique cluster. As for partitional clustering, the process reiterates until the entire tree is created, or the number of clusters has met the threshold θ .

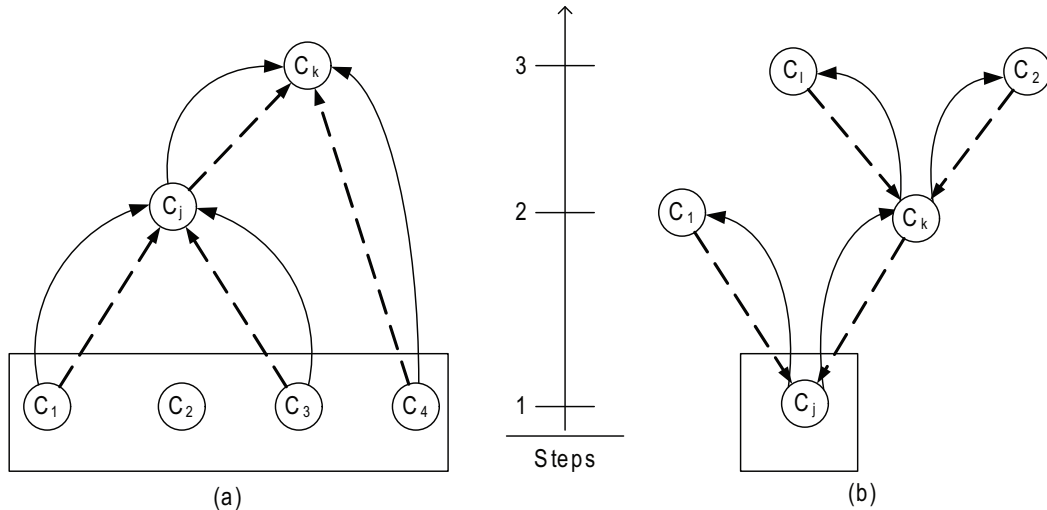


Figure 6.1: Illustration of the first three clustering steps for the (a) partitional and (b) agglomerative clustering algorithm.

Figure 6.1 illustrates the first three steps of the 6.1(a) partitional and 6.1(b) agglomerative clustering algorithm. The obvious difference between these two approaches is that partitional clustering algorithms are top-down approaches, while agglomerative clusterings are bottom-up approaches. The main advantage of the partitional algorithms is their low complexities, which allow them to cluster millions of elements. However, partitional algorithms can suffer from local minima, and are dependent on the input order of the items. On the other hand, in general, agglomerative algorithms give better clustering solutions than partitional algorithms. However, [Zhao and Karypis, 2005] have shown that for clustering document data sets, partitional clusterings always led to better clustering solutions than agglomerative ones. The main disadvantage of agglomerative clusterings is their complexities. In the first step of the algorithm, all pairwise similarities must be computed, leading to a complexity of $O(n^2)$.

Concept-based clustering

The most famous (incremental) conceptual clustering algorithm is *COBWEB*, which was introduced by [Fisher, 1987]. Contrary to the first two clustering algorithms, items need to be represented by a set of attribute-value pairs. For example, Table 6.1 illustrates this principle on three items that have three attributes: *BodyCover*, *HeartChamber*, and *BodyTemp*.

| Name | Body Cover | HeartChamber | BodyTemp |
|--------|------------|--------------|-------------|
| mammal | hair | four | regulated |
| bird | feathers | four | regulated |
| fish | scales | two | unregulated |

Table 6.1: Example of attribute-value pairs for COBWEB.

Given this representation, a node (class) in the tree represents the probability of the occurrence of each attribute-value pair for all the instances of that node. Thus, the root node

represents all the possible attribute-value pairs defined in the system, and nodes become more specific when traveling down the tree.

For each item to classify, COBWEB will incrementally incorporate it to the classification tree by descending the tree along an appropriate path, updating counts along the way, and performing one of the four operators at each level. These operators are as follows.

- *add* - adds the item to an existing class,
- *create* - creates a new class for the item,
- *merge* - merges two classes, and
- *split* - splits a class into several ones.

As items are added incrementally, the ordering of the initial input can lead to different classification results. To reduce this problem, [Fisher, 1987] uses the operators *merge* and *split*. Furthermore, node merging and splitting are roughly inverse operators, and allow COBWEB to move bidirectionally through a space of possible hierarchies.

The choice of the operator will be guided by a *category utility* function that computes the rewards of traditional virtues held in clustering, i.e.: similar objects should appear in the same class, while dissimilar ones should be in different classes. Formally, given a partition of n classes $\{C_1, C_2, \dots, C_n\}$, Equation 6.1 shows the category utility function used by COBWEB, where $P(C_k)$ is the size of the class C_k as a proportion of the entire data set, $P(A_i = V_{i,j})$ is the probability of attribute A_i taking on the value $V_{i,j}$, and $P(A_i = V_{i,j} | C_k)$ is the conditional probability of $A_i = V_{i,j}$ in the class C_k .

$$\frac{\sum_{k=1}^n P(C_k) \left[\sum_i \sum_j P(A_i = V_{i,j} | C_k)^2 - \sum_i \sum_j P(A_i = V_{i,j})^2 \right]}{n} \quad (6.1)$$

Thus, the category utility function is in fact a trade off between intra-class similarity and inter-class dissimilarity of objects, where objects are described in terms of attribute-value pairs like those of Table 6.1. Detailed explanation of the computation of all these probabilities can be found in [Fisher, 1987].

One of the main advantages of COBWEB over the partitional and agglomerative clustering algorithms is that it allows new items to be added incrementally to the classification tree, without recomputing the entire tree. Note also that the process is bidirectional, which means that a node that has been created can be merged again later on. Second, conceptual clustering uses probabilistic descriptions of concepts, rather than distances. Furthermore, the operators merge and split guarantee homogeneity of the content of the class. However, COBWEB has some known problems. First, the classification tree is not height-balanced which leads to space and time complexity to degrade dramatically. Second, the overall complexity of COBWEB is exponential to the number of attributes, as the category utility function requires analyzing all the attribute-value pairs.

Despite these problems, and because most authors assume that the number of attributes is small, COBWEB is becoming increasingly popular in the Semantic Web. For example, [Clerkin et al., 2001] proposed to use COBWEB directly on the user-item matrix R to build meaningful hierarchical ontologies of songs. To do this, they consider each item as an

attribute, and the ratings assigned to each item is transformed in the feature value *good* if it had a rating superior or equal to 4, or *bad* otherwise. In [Seo and Ozden, 2004], Seo and Ozden have used COBWEB to generate the ontology from files' description in order to perform ontology-based file naming.

6.3 Learning a set of hierarchical taxonomies

Distance based clustering algorithms cluster items based on the optimization of a distance function, or inversely a similarity function. Thus, the first task is to define a function that is capable of comparing two items i and j from the user-item matrix R . Sections 2.4.1 and 2.4.2 introduced the three most common similarity functions used by collaborative filtering: Pearson, cosine, and adjusted cosine; which are respectively defined in Equations 2.19, 2.20 and 2.24. These functions transform the user-item matrix R into an item-to-item matrix S , where $S_{i,j}$ is the similarity between items i and j . As this dissertation uses rated items to model user's preferences, a similarity metric will also be used to generate the similarity matrix S from the user's rated item. Furthermore, it has previously been argued that the best similarity function is the adjusted cosine, as it takes into account the difference in rating of each user's profile. Notice that this is achieved by subtracting from each user's ratings its average rating \bar{R}_u . Thus, this dissertation will use Equation 2.24 to measure the similarity between two items. Formally, the process for generating the similarity matrix S is as follows. First, the rated items are extracted from the users' preference profiles and aggregated into a user-item matrix R . Then, Equation 2.24 is used to compute the pairwise similarity between all the items in R . Finally, all these pairwise similarities are stored in the item-to-item similarity matrix S .

Next, *criteria functions* or distance functions need to be defined in order to optimize the allocation of an item to a given cluster [Zhao and Karypis, 2005]. Table 6.2 shows the criteria functions that will be used for generating the ontologies, where k is the number of clusters to consider, n_r denotes the number of elements in cluster r , C_r is the r^{th} cluster, C_r^t is the centroid of the r^{th} cluster, and $sim(i, j)$ is the similarity between items i and j .

| | | |
|---|-----------------|---|
| 1 | \mathcal{I}_1 | $maximize \sum_{r=1}^k \frac{1}{n_r} \left(\sum_{i,j \in C_r} sim(i, j) \right)$ |
| 2 | \mathcal{I}_2 | $maximize \sum_{r=1}^k \sum_{i \in C_r} sim(i, C_r^t)$ |
| 3 | \mathcal{E}_1 | $minimize \sum_{r=1}^k n_r sim(C_r^t, C)$ |
| 4 | \mathcal{G}_1 | $minimize \sum_{r=1}^k \frac{cut(C_r, C - C_r)}{\sum_{i,j \in C_r} sim(i, j)}$ |
| 5 | \mathcal{H}_1 | $maximize \frac{\mathcal{I}_1}{\mathcal{E}_1}$ |
| 6 | \mathcal{H}_2 | $maximize \frac{\mathcal{I}_2}{\mathcal{E}_1}$ |
| 7 | <i>slink</i> | $\max_{i \in C_i, j \in C_j} sim(i, j)$ |
| 8 | <i>clink</i> | $\min_{i \in C_i, j \in C_j} sim(i, j)$ |
| 9 | UPGMA | $maximize \frac{1}{n_i n_j} \sum_{i \in C_i, j \in C_j} sim(i, j)$ |

Table 6.2: Criteria functions used by distance-based clustering algorithms to merge and split clusters.

During partitional clustering, a distance function d is required in order to assign each item from cluster C_j to the cluster C_i that optimizes d . The partitional algorithms will use the criteria \mathcal{I}_1 , \mathcal{I}_2 , \mathcal{E}_1 , \mathcal{H}_1 , and \mathcal{H}_2 as distance function. \mathcal{I}_1 and \mathcal{I}_2 are *internal* criteria functions that focus on producing a clustering solution that optimizes the function over the items of each cluster individually. The essential difference between \mathcal{I}_1 and \mathcal{I}_2 is that the former maximizes the average pairwise similarities between all the items assigned to each cluster, while the latter represents each cluster by a center of gravity, known as *centroid*, and then looks at the similarity between the items and this centroid. Notice that \mathcal{I}_2 is in fact equivalent to the very popular *K-Means* algorithm, with K set to 2. However, \mathcal{E}_1 is an *external* criterion function as it looks on how the various clusters are different from each other, and tries to minimize the cosine between the centroid of each cluster. \mathcal{G}_1 is a *graph based* criterion function that models the items as a graph (the node corresponds to an item, while an edge between a pair of nodes measures the similarity between each of these nodes), and uses a variant clustering quality measure (the min-max cut, [Ding et al., 2001]). Finally, \mathcal{H}_1 and \mathcal{H}_2 are *hybrid* criteria functions that simultaneously optimize multiple individual criteria functions. Many techniques have been proposed for selecting which clusters to choose next for bisection, ranging from random policy, to size analysis. In order to obtain a more natural hierarchical solution, [Zhao and Karypis, 2005] proposed to choose the cluster among the k possible choices as the one that leads to the $k + 1$ clustering solution that optimized one of the above criteria functions.

The key criterion in agglomerative clustering is the function used to merge a pair of clusters. The three criteria functions commonly used today are: *slink*, *clink* and *UPGMA*. The *single-link*, *slink*, and *complete-link*, *clink*, criteria functions both compute the similarities by considering a pair of items in the different clusters. The main difference lies in the fact that the *single-link* considers the closest elements of the two clusters, while the *complete-link* looks at the furthest pair. However, these criteria functions do not perform well in practice because they only use limited information. The *UPGMA* criterion function overcomes these problems by measuring the similarity of two clusters as the average of the pairwise similarity of the items in each cluster. Besides these three functions, the first six functions in Table 6.2 can also be used for selecting which clusters to merge [Zhao and Karypis, 2005].

Formally, Algorithm 7 generates a set of 15 taxonomies as follows. First, a matrix S , and a set Λ is initialized. The matrix S will contain all the item-to-item similarities, while the set Λ will store the learnt taxonomies. In step 2, the user-item matrix R is generated from all the users' preference profiles, while step 5 computes the matrix S using the adjusted cosine similarity function (Equation 2.24). Using $S_{i,j}$ as $sim(i, j)$, and a threshold θ as the number of leaf clusters, the algorithm generates 15 distinct hierarchical trees using the partitional (step 7) and agglomerative clustering (step 9) algorithms introduced respectively in subsection 6.2. Note that if the threshold parameter θ is set to any value less than the total number of items to be clustered, then some leaf clusters will contain more than one item. Finally, the algorithm terminates by returning the 15 hierarchical trees. The eCatalog ontological models can then be created from these clustering trees by using Algorithm 1.

Algorithm 7: *Learning a set of 15 distinct taxonomies, each having a hierarchical structure.*

Inputs: All the users's preference profile $UPPs$, and the number of leaf clusters θ , desired.

Outputs: A set Λ containing 15 different taxonomies with hierarchial structure.

Functions: $createUserItemMatrix(UPPs)$ creates the user-item matrix R from $UPPs$, $acSim(i, j)$ computes pairwise adjusted cosine similarities between item i and j , and the criteria functions from Table 6.2.

```

1  $S \leftarrow \phi, \Lambda \leftarrow \phi$ 
2  $R \leftarrow createUserItemMatrix(UPPs)$ 
3 forall  $i \in R$  do
4   | forall  $j \in R$  do
5   |   |  $S \leftarrow acSim(i, j)$ ;
6   |   end
7   end
8 for criteria function 1 to 6 do
9   |  $\Lambda \leftarrow \Lambda \cup$  tree generated by partitional clustering.
10  end
11 for criteria function 1 to 9 do
12  |  $\Lambda \leftarrow \Lambda \cup$  tree generated by agglomerative clustering.
13  end
14 return  $\Lambda$ 

```

6.3.1 Selecting the best learnt eCatalog ontological model

It is a naive assumption to believe that all users behave in the same way. Moreover, each person perceives differently the world they live in. This dissertation believes that the accuracy of the recommendation can be increased by carefully selecting the eCatalog ontological model, rather than using the same one for all the users.

Moreover, in order to minimize the propagation error, the inference process defined in Equation 5.7 needs to find the closest concept with a score to the one we want to infer the score from. Furthermore, as the inference is done from only one concept, the selection of the closest concept is fundamental. As users tend to like similar items, this implies that these items will be represented by concepts in the ontology which tends to be close to each other. Thus, if the concepts that represent the items liked by the user are too distant from the disliked ones, then the inference will introduce a bias towards the liked concepts (remember that at least 70% of the elicited items are liked by the user).

Following these observations, this dissertation proposes to select the eCatalog ontological model based on the user's preference. This is achieved by choosing the ontology that minimizes the distance between the liked concepts and the disliked ones. Formally, Algorithm 8 is a two step process that is based on this idea. First, it selects a subset of ontological model that it thinks will perform the best, and then selects the model that minimizes the distance between liked and disliked concepts out of the selected ontologies. The first stage

of the algorithm simply limits the search of the right model as the computation of distances is computationally expensive. The second stage then looks for the optimal model.

Algorithm 8: *Selecting the best eCatalog ontological model from a set of possibilities eCOMs for user u.*

Inputs: The set of eCatalog ontological model *eCOMs*, the user's preference profile *UPP* ontologies Λ , and the user's learning set *LS*.

Outputs: An eCatalog ontological model that best matches the user's preferences.

Functions: *buildOntologicalProfile*(*UPP*, *eCOM*) creates the user ontological profile using Algorithm 3.

```

1 Split UPP into two distinct sets: 90% in Knowledge and the remaining in Test.
2 forall  $\lambda$  in eCOMs do
3   | knowUOP  $\leftarrow$  buildOntologicalProfile(Knowledge,  $\lambda$ )
4   | testUOP  $\leftarrow$  buildOntologicalProfile(Test,  $\lambda$ )
5   | Predict the concepts' scores in testUOP based on knowUOP using Algorithm 5
6   | preci(testUOP)  $\leftarrow$  precision of the generated score of testUOP
   end
7 besteCOMs  $\leftarrow$  the eCatalog models with the best preci(testUOP)
8 forall model  $\in$  besteCOMs do
9   | UOP  $\leftarrow$  buildOntologicalProfile(UPP, model)
10  | Liked  $\leftarrow$  {c | c  $\in$  UOP  $\wedge$  S(c)  $\geq$  0.8}
11  | Disliked  $\leftarrow$  {c | c  $\in$  UOP  $\wedge$  S(c) < 0.8}
12  | Dist(model) =  $\frac{\sum_{n \in Liked, m \in Disliked} D(n, m)}{|Liked| |Disliked|}$ 
   end
13 return eCOM | argeCOM  $\in$  besteCOMs min(Dist(eCom))

```

Algorithm 8 starts by splitting the user's preference profile *UPP* into two distinct sets: *Knowledge* and *Test*. *Knowledge* will be used for learning the scores, and will contain 90% of the data in *UPP*. The *Test* set will contain the remaining ratings, and will be used to test the scores learnt on *Knowledge*. For each of the 15 eCatalog ontological models, steps 3 and 4 build the user's ontological profiles for the items contained respectively in *Knowledge*, and *Test*. Using Algorithm 5, step 5 infers the score of all the concepts in *testUOP* using the concepts in *knowUOP*. Then, the precision of the inferred scores are computed based on the real score found in *TestUOP*, where the precision is defined as the ratio of correct items found by the size of *Test*. In step 7, the eCatalog ontological models that achieve the best precision are selected. This selection process is as follows. If there are at least two models with the highest precision value, then these two models are returned. Otherwise, it is the best eCatalog ontological model along with the second best that are selected. This selection process ensures that there are always at least two models for the rest of the algorithm. For each selected model, step 12 uses Equation 5.21 to compute the distance between the concepts liked by the user and the disliked ones, where *n* and *m* are concepts respectively from the set of concepts *Liked* that are liked by the user, and from the set of concepts *Disliked* that she disliked. Note that step 10 and 11 use the 0.8 threshold

to classify a concept as liked or disliked, which corresponds to a rating of 4. Finally, step 7 returns the eCatalog ontological model that has the smallest overall distance.

6.4 Learning multi-hierarchical taxonomies

This dissertation believes that the recommendation accuracy can be further be improved if the search space is slightly increased.

When using classical distance-based clustering algorithms to generate ontologies, all the implicit features between a concept and its sub-concept are stored in a single edge. This could potentially limit the concept representation, and thus limit OF's inference process. Moreover, the hierarchical clustering algorithms used in Algorithm 7 always select the best cluster to merge/split based on the optimization of one of the criterion function in Table 6.2. Thus, it ignores other possible suboptimal candidates.

In Chapter 8, experimental results show that, on average, it is the agglomerative clustering with the complete-link criterion function that achieves the best result. Algorithm 9 extends this algorithm in order to build multi-hierarchical taxonomies. An illustration of one iteration of Algorithm 9 is presented in Figure 6.2.

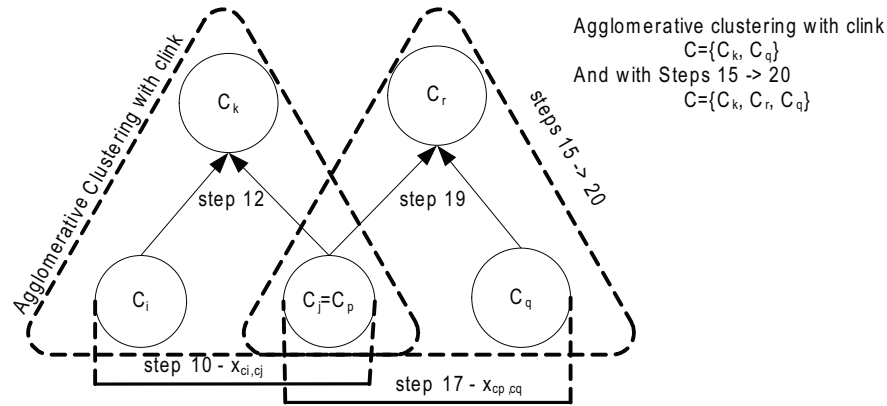


Figure 6.2: Illustration of the multi-hierarchical structure generated by Algorithm 9.

As for Algorithm 7, the first 5 steps of Algorithm 9 simply create the item-to-item similarity matrix from the users's preference profile. The steps 6 to 14, and the step 21 are in fact the classical agglomerative clustering with the complete-link criterion function, where the complete-link criterion function is being defined in step 10. Given a coefficient $\varphi \in [0..1]$ as input parameter, step 15 computes a window size of acceptable clusters based on the value of the current criterion value and φ . Given this window size, step 17 looks at all the possible pairs of clusters that can be made with one of the merged clusters C_i or C_j , and step 18 checks if its criterion value is within the window size. For each pairs within the window, step 19 merges them into a unique cluster C_r , while step 20 adds C_r to the list of open clusters C . Notice that x_{C_p, C_q} cannot be bigger than x_{C_i, C_j} as the pair $C_i C_j$ optimizes the criterion function in step 10. As cluster C_p has already got a parent (i.e.: C_k), cluster C_r becomes the second parent of cluster C_p . Finally, step 22 returns the DAG structure in C .

Algorithm 9: Learning a Multi-Hierarchical ontology with a window size φ and threshold cluster θ for user u .

Inputs: All the users's preference profile $UPPs$, and the number of leaf clusters, coefficient φ , and θ .

Outputs: An ontology with a multi-hierarchical structure.

```

1  $S \leftarrow \phi, C \leftarrow \phi$ 
2  $R \leftarrow createUserItemMatrix(UPPs)$ 
3 forall  $i \in R$  do
4   | forall  $j \in R$  do
5   |   |  $S \leftarrow acSim(i, j);$ 
6   |   end
7   end
8   Assign each item  $i$  to its own cluster  $C_i$ , and make  $C = C \cup C_i$ 
9   while  $|C| > \theta$  do
10    |  $X \leftarrow \phi$ 
11    | forall  $C_i, C_j$  in  $C$  do
12    |   |  $X \leftarrow X \cup (x_{C_i, C_j} = \min_{i \in C_i, j \in C_j} sim(i, j))$ 
13    |   |  $x_{C_i, C_j} \leftarrow max(X)$ 
14    |   |  $C_k \leftarrow merge(C_i, C_j)$ 
15    |   |  $C_m \leftarrow \{C_i, C_j\}$ 
16    |   |  $C \leftarrow C / C_m$ 
17    |   |  $windowMinVal \leftarrow x_{C_i, C_j} - \varphi x_{C_i, C_j}; X \leftarrow \phi$ 
18    |   | forall  $C_p$  in  $C_m$  and  $C_q$  in  $C$  do
19    |   |   |  $x_{C_p, C_q} \leftarrow \min_{p \in C_p, q \in C_q} sim(p, q)$ 
20    |   |   | if  $x_{C_p, C_q} > windowMinVal$  then
21    |   |   |   |  $C_r \leftarrow merge(C_p, C_q)$ 
22    |   |   |   |  $C \leftarrow C \cup C_r$ 
23    |   |   |   end
24    |   |   end
25    |   end
26    |  $C \leftarrow C \cup C_k$ 
27  end
28 end
29 return  $C$ 

```

Experimental results show that Algorithm 9 is capable of increasing the recommendation accuracy, when φ is in $[0, 0.4]$. However, the increase of accuracy is at a cost of computational complexity, as more edges are created. Thus, a tradeoff has to be done between computational complexity and recommendation accuracy.

Chapter 7

Architecture of ontology filtering

Previous chapters defined various algorithms for building preference profiles and eCatalog ontological profiles, inferring missing preferences from an incomplete model, and learning a set of taxonomies from users' past experience. All these algorithms are key elements of the ontology filtering recommender system, and this chapter will define the architecture that puts all the pieces of the puzzle back together.

Concretely, this chapter describes in detail the three tier architecture used by ontology filtering. As this dissertation specifically focuses on recommender systems algorithms, the emphasis will be put on the application layer. However, the presentation and data layers will also be described, but in less details as it is out of the scope of this thesis.

7.1 Introduction

As introduced in Chapter 1.2, traditional recommender systems use a 3-tier architecture. As shown in Figure 7.1, the 3-tier layer is in fact a client-server model composed of the following layers:

- **The Presentation Layer:** contains the graphical user interface, GUI, that allows a person and system to interact with each other. In the eCommerce environment, web browsers such as Internet Explorer are the most common form of GUI. Furthermore, these web browsers are located on the customer's device, which can be anything from a computer to a mobile phone.
- **The Application Layer:** is the layer that is in charge of processing the user's preferences, and making the recommendations. The application layer is made of two components: the preference manager and the recommendation engine. The former interacts with a user in order to extract and maintain the user's preferences, while the later makes use of the collected preferences to generate the recommendations.
- **The Data Layer:** is responsible for storing and accessing the data required by the application layer. Typically, this layer is made of two databases. One for storing the user's preferences, and another one for storing the items making the eCatalog.

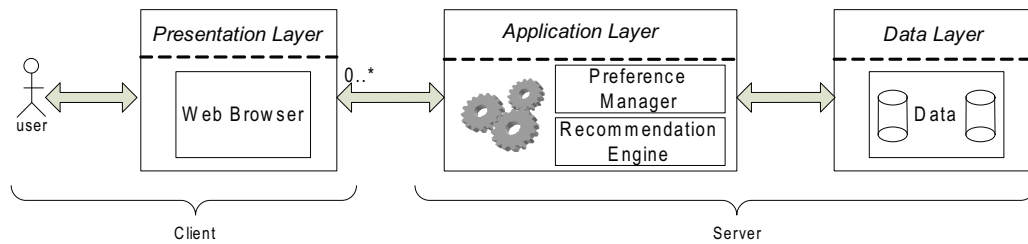


Figure 7.1: The three tier architecture with a thin-client.

A common dilemma associated with the client-server is the extent to which the client is used in the recommendation. For example, a *thin client* simply elicits the user's preferences and displays the recommendations, while a *fat client* will do the same as a thin client but will also be involved in the computation of the recommendations. For example, java applets are widely used to make fat clients, as they allow a client to run java-code on their machine. The main advantage of fat clients is that they distribute the computation over the clients, which increases the scalability of the system, while reducing the cost of the computer infrastructure. Moreover, the distribution of the recommendations allows to increase the user's privacy. However, thin clients are seen as better solutions for the following reasons:

- *Cost* - Thin clients are much cheaper to conceive as they simply focus on displaying the results, without considering expensive distribution steps. Moreover, they reuse existing free technology such as HTML and PHP, which further reduces the costs.
- *Easier to manage* - Thin clients are not involved in the computation of the results, which simplifies the maintenance of the system as a whole. Imagine a situation where a bug is found in the computation of the recommendation. With a thin client, fixing the problem usually only requires patching the server, which can be done without the user's intervention and knowledge. If some of the computation is done by the clients, and if you have 1 million customers, then you will need to patch 1 million applications.
- *Easier to use* - Users do not need to install third party softwares, but can simply use their web browsers to display the results. Furthermore, and to protect the user's privacy and security, today's firewalls and strict security rules in web browsers only let users perform primitive tasks on their computer.

As with most recommender systems, ontology filtering has been designed on the 3-tier architecture with a thin client. However, as shown later in this chapter, ontology filtering allows the computation of the recommendations to be easily distributed over the clients. Finally note that, to illustrate the ontology filtering process, Appendix C contains snapshots of a full recommendation cycle for jokes.

7.2 The architecture

Figure 7.2 shows the architecture of the ontology filtering recommender system. As it is based on the 3-tier paradigm with thin clients, it contains a presentation layer, an application layer and a data layer, which are described below.

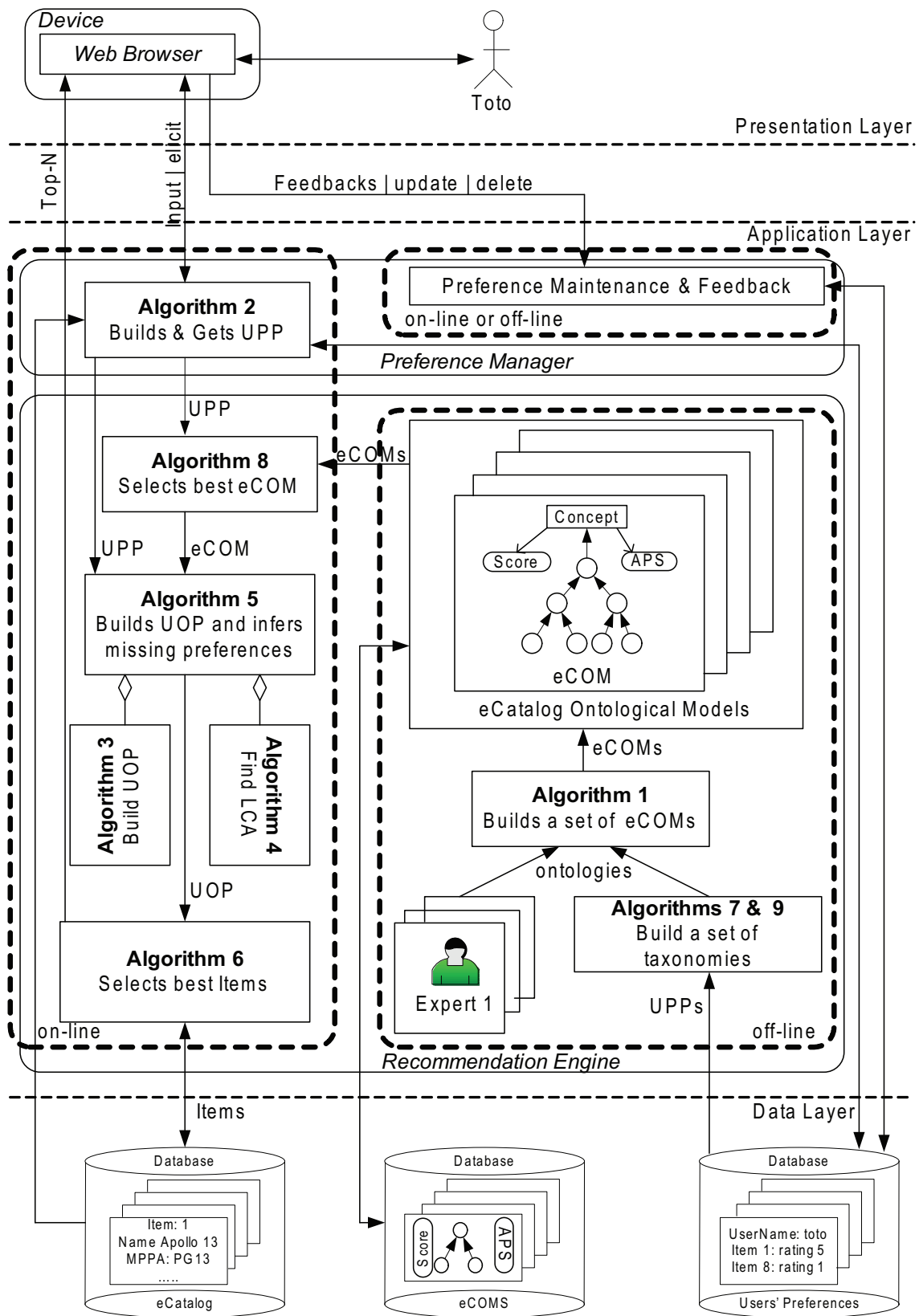


Figure 7.2: The architecture of the ontology filtering recommender system.

7.2.1 The presentation layer

The presentation layer is composed of two elements: the device and the user. The user represents the person who wants some recommendations, while the device is the computer system that allows the user to interact with the application layer.

More specifically, the user interacts through her device's web browser using three fundamental operations: clicking, selecting, and setting the value of predefined fields. These operations allow the user to manage her preferences, and select which items among the recommendations she likes. Note that the first two operations require very low cognitive thinking from the user, while the third operation allows expert users who have better understanding of the domain to enter more precise preferences.

Find a joke

Before we recommend you stuff, we need to know more about your taste in jokes

Please rate 2 jokes:

Either:

(a) Enter a joke you know:

Joke

How much did you like it? Awful (★☆☆☆☆) Really bad Not good OK Excellent (★★★★★)

OR

(b) Rate the following joke:

Joke no: 86 (step 1 - 2)

A neutron walks into a bar and orders a drink. "How much do I owe you?" the neutron asks.

The bartender replies, "for you, no charge."

How much did you like it? Awful (★☆☆☆☆) Really bad Not good OK Excellent (★★★★★)

Figure 7.3: Snapshot of the preference elicitation interface used in ontology filtering.

Figure 7.3 illustrates the web page that allows a user to state her preferences. In this figure, the user is being asked to give her preferences on 2 jokes by rating either a known joke, or a randomly selected one. This figure also illustrates the three fundamental operations a user can do. For example, a user can select which of the two approaches they want to use for expressing preferences, and must also select the rating assigned to a joke. If the user decides to enter a known joke, then the user must enter either the name or part of the content of the joke in the predefined field *Joke*. Finally, once the user has finished, she goes to the next page by clicking on the *Next >>* button.

7.2.2 The application layer

This is the core layer that contains all the logic of the recommender system. This layer is in turn composed of the following sub-components:

- *The preference manager* is responsible for building the user's preference profile through elicitation questions, gathering the feedback from the users, and maintaining their preferences. Recall that in ontology filtering, all the preferences (elicited or feedbacks) are explicit ratings on items that range in the interval $[1, 5]$.
- *The recommendation engine* handles the eCatalog ontological models, and uses them, along with the user's preference profile, to generate the recommendations.

Moreover, the behavior of the application layer depends on two states: *on-line*, or *off-line*. These states reflect whether a user is connected to the application layer and requesting some recommendations, or not. If she is, then the application layer is in an on-line state, otherwise it is off-line. The main reason for differentiating these two states is to allow the system to perform some pre-processing of the data in order to speed up the recommendations. Note that the preference maintenance and feedbacks processes are state independent, as it can be done independently from a recommendation. For example, a user can give feedbacks weeks after getting the recommendations. Similarly, a user can modify previously stated preferences at any point in time.

The off-line phase

Ontology filtering uses ontologies to model the items of an eCatalog. These eCatalog ontological profiles are constructed using taxonomies that are obtained from a combination of the following ways:

Manually - In this situation, a group of experts sits around a table, and comes to a consensus about the concepts modeling the universe of discourse. For some simple domains that do not change too frequently, it is feasible for experts to construct and maintain such ontologies.

Automatically - However in eCommerce environments, it is usually not realistic to assume the existence of ontologies, or that experts can create them. To overcome this problem, ontology filtering proposes to learn them ontologies automatically, and without any expert intervention using two algorithms: Algorithms 7 and 9. Both algorithms create taxonomies from the users' preference profile, but there are two fundamental differences between them. First, the former algorithm creates a set of 15 distinct taxonomies, while the latter only produces one. Second, Algorithms 7 produces taxonomies with hierarchical structures, while the latter contains a single one but with a multi-hierarchical structure.

Algorithm 1 is then used over all these taxonomies in order to produce a set of eCatalogs ontological models. Once these models have been created, a copy is sent to the data layer for persistent storage, while another remains in memory for the recommendations (i.e.: the on-line state).

Theoretically, the ontological models should be regenerated as soon as a user submits new ratings, as these ratings modify the item-to-item matrix. As eCatalogs contain millions of items, it is unfeasible to regenerate the ontologies every time a user submits a new rating. However, clustering algorithms allow the ontology learning phase to be performed daily and in a distributed fashion. For example, a simple distributed algorithm would be to distribute the construction of each ontology to a different server.

The on-line phase

Once at least one eCatalog ontological model is available, ontology filtering can start predicting items to the users. Whenever a user wants some recommendations, the on-line recommendation process is as follows:

1. The user's preference profile is built using Algorithm 2. There are two situations to consider, which depend on whether or not the preference profile exists. If the user is new to the system, then Algorithm 2 elicits at least 5 ratings from the user. Otherwise, it checks in its database to see if the profile exists, and if it is valid. If it is, then the profile is loaded and the process continues to step 2. Otherwise it elicits the missing information. Checking whether an existing profile is valid or not seems useless, but it is vital as users have the possibility, through the preference maintenance process, to modify and delete any or all of their preferences.
2. Using the user's preference profile UPP , Algorithm 8 goes through the set of available ontological models, and selects the one that best matches the user's preference profile. Recall that the best ontological model, $eCOM$, is the one that minimizes the semantic distance between the concepts the user liked and disliked.
3. Algorithm 5 then uses the best model found in step 2, and starts by building the user's ontological profile. Then, the missing scores of each concept are inferred from the closest known score using the inference mechanism defined in Equation 5.8. This algorithm allows the system to build a complete user's ontological profile UOP , where each concept has a score that represents how much the user is going to like the concept's instances.
4. Finally, Algorithm 6 generates a set of N recommendations, and returns them to the user. The recommendation is made by taking a list of possible candidate $Item$, and looking at the hybrid score of the concept to which these items are instanced. Thus, the N returned items are those whose concepts have the highest hybrid score, where the hybrid score is computed using Equation 5.22. Note that the personalization coefficient ρ is initially set to 0.5, but it can be freely changed by the user.

Once the items have been recommended, the user has the possibility to give feedbacks, which helps the system updates the user's preference profiles.

7.2.3 The data layer

The data layer is the memory of any recommender system. Traditional recommender systems typically have two databases. The first one stores the items contained in the eCatalog, while the second one contains the preference profiles of all the users. Ontology filtering adds a third database that will store the eCatalog ontological models.

Once the eCatalog ontological models have been created, a copy is sent to the data layer in order to be inserted in the database. This allows the system to retrieve the models in case the system crashes, which is much faster than re-learning them. Another advantage of using a database is the possibility to update the eCatalog ontological models while the recommendation engine is running. Thus, it increases the recommender system availability, and allows the ontological models to be updated whenever required.

Obviously, the eCatalog ontological models can not be stored as such in the database, but require some transforming before being inserted in the database. Figure 7.4 proposes a

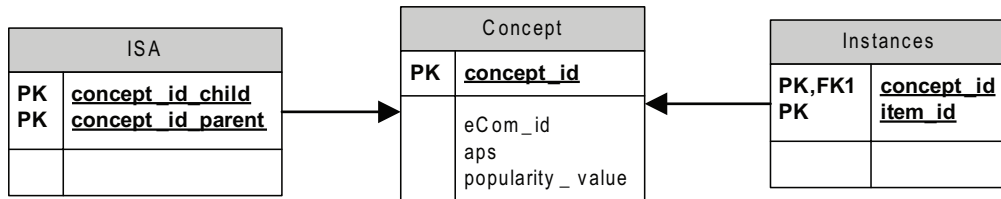


Figure 7.4: A possible entity-relationship schema for the eCatalog ontological model.

simple entity-relationship schema made of three tables. These tables can store all the content and structure of the eCatalog ontological models. The schema contains the following tables:

- *Concept* represents a concept in the eCatalog ontology model, $eCOM_{id}$, and is identified by *concept_id*. The fields *aps*, and *popularity_value* respectively store the a-priori score, and the popularity value of a concept. Note that it not necessary to store the score of a concept as it is user dependant, and initially set to 0.0.
- *ISA* models the inheritance relationships in the eCatalog ontological model. Take for example Figure 7.5, where concept x is the child of concept y . Thus, table *ISA* will have a field $\langle concept_id_child = x, concept_id_parent = y \rangle$.

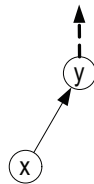


Figure 7.5: Example of a hierarchical relation in the eCOM, where concept x is the child of concept y .

- *Instances*, as its name indicates, contains all the instances of a concept. Note that each instance is identified by an *item_id*

Chapter 8

Experimental results

Experimental results are now presented that validate the ontology filtering recommender system. These results are presented in two phases. First, the similarity metric OSS is tested on WordNet and on the GeneOntology, which validate the inference mechanism. In the second phase, the ontology filtering recommender system as a whole is tested on two data sets: Jester and MovieLens. These two sets contain real user ratings respectively on jokes and movies. Experimental results on both data sets show that ontology filtering significantly increases the accuracy of the recommendations, while being more scalable.

These results tend to confirm that the user's preferences follow the topology of an ontology, and that this topology can successfully be used to infer missing preferences. We also believe that another reason why collaborative filtering fails to correctly predict items when few ratings are known is the unrestricted search space. The experiments shows that, even when very few ratings are known, ontology filtering is capable of achieving high accuracy. This suggests that the ontology can successfully restricts the search space.

8.1 Validation of the model

Testing the inference mechanism is a very complex task as it requires finding a big enough set of real users, and asking them how much they like an item given another one. Moreover, the inference mechanism relies on an ontology, which is also very hard to construct.

Fortunately, Chapter 5.4.4 has shown that a similarity function can be derived from the inference mechanism. Moreover, semantic similarity has been heavily studied in computer science and linguistic. From this research, a well established evaluation procedure on the WordNet ontology has emerged.

WordNet, with its 117'000 concepts, is one of the biggest ontologies in the world (Appendix D.1). The goal of WordNet is to develop a system that would be consistent with the knowledge acquired over the years about how human beings process and see the English language. In the case of hyponymy, psychological experiments¹ revealed that individuals can access properties of nouns more quickly depending on when a characteristic becomes

¹<http://en.wikipedia.org/wiki/WordNet>

a defining property. That is, individuals can quickly verify that canaries can sing because a canary is a songbird (only one level of hyponymy), but require slightly more time to verify that canaries can fly (two levels of hyponymy) and even more time to verify canaries have skin (multiple levels of hyponymy). This suggests that humans store semantic information in a way that is much like WordNet, because we only retain the most specific information needed to differentiate one particular concept from similar concepts.

Ontology filtering assumes that users' preferences follow an ontology, and that such ontology can be used to infer missing preferences. From the inference mechanism, a similarity function called OSS has been derived that can compute the pairwise similarity between any pair of concepts in the ontology. This similarity function uses a-priori scores of concepts that behave the same way as users see features in items.

Following this, the first experiment will look at the correlation between the similarities computed with *OSS* and real human evaluation. The experiment shows that OSS correlates well with human evaluation (over 91%), and significantly outperforms state of the art similarity metrics. A similar experience has also been done on the GeneOntology, and results consolidate the previous findings by showing that OSS significantly outperforms existing metrics in every situations.

8.1.1 Experiment I - WordNet

When Resnik introduced the node-based similarity approach [Resnik, 1995], he also established an evaluation procedure that has become widely used ever since. He evaluated his similarity metric by computing the similarity of word pairs using the WordNet ontology, and then considered how well it correlated with real human ratings of the same pairs. More information about the WordNet ontology can be found in Appendix D.1.

An experiment by [Miller and Charles, 1991] provided appropriate human subject data for this kind of experiment. In their study, 38 undergraduate subjects were given 30 pairs of nouns that were chosen to cover high, intermediate, and low levels of similarity. These subjects were asked to rate the *similarity of meaning* for each pair in a scale from 0 (no similarity) to 4 (perfect synonyms). The average rating for each pair represents a good estimate of how similar two words are, according to human judgments.

Resnik reproduced the Miller&Charles experiment on a subset composed of 28 word pairs, and found very close correlation between the human judgments obtained by Miller and Charles. Only 28 word pairs were considered in Resnik's experiment, because WordNet 1.4 did not contain all of the word pairs.

| | Edge | Leacock | Resnik | Lin | Jiang | OSS |
|-------------|-------|---------|--------|-------|-------|--------------|
| Correlation | 0.603 | 0.823 | 0.793 | 0.823 | 0.859 | 0.911 |

Table 8.1: Correlation of various similarity metrics with human judgements collected by Miller and Charles on WordNet 2.0.

This dissertation reproduces the exact experiment proposed by Miller and Charles, and uses WordNet version 2.0. Furthermore, to correctly benchmark the *OSS* similarity metric, all of the similarity metrics introduced in Section 5.4.1 have also been implemented and

tested. The correlations between various metrics and the human ratings are displayed in Table 8.1.1.

These results show that *OSS* achieves over 91% correlation with real user ratings, and clearly demonstrate significant benefit over earlier approaches ($t\text{-obs} \simeq 3.28$ and $p\text{-value} < 0.02$). Note that the hybrid approach defined by [Jiang and Conrath, 1997] performed better than both edge based and node-based techniques, but the improvement over the information based approach was not statistically significant ($t\text{-obs} = 1.46$ and $p\text{-value} \simeq 0.08$). As expected, the edge based approach is the worst performing metric as it supposes that the edges represent uniform distances, which is obviously not true in WordNet.

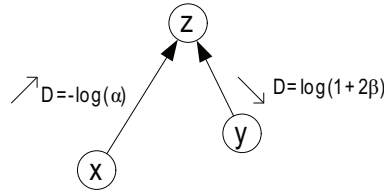


Figure 8.1: Illustration of the upwards and downwards distance in OSS.

A major difference between OSS and existing similarity metric is the fact that OSS is asymmetric, which means that the similarity between concepts a and b is different from the similarity between b and a . To test this aspect, different combinations of the coefficients α and β have been tested in order to test the upward and downward propagation. Note that the order of the word pairs given to OSS was exactly the same as the one given to the human subjects in the Miller and Charles experiment. As expected, Table 8.2 shows that the best correlation is obtained when using α going up and $1 + 2\beta$ going down. As mentioned earlier, these coefficients render the metric asymmetric. In fact, experiments showed that the upwards distance is up to 15 times greater than the downwards distance when concepts are very dissimilar.

| | | | | |
|---------------|--------------|----------|--------------|--------------|
| coefficient ↗ | α | α | $1 + 2\beta$ | $1 + 2\beta$ |
| coefficient ↘ | $1 + 2\beta$ | α | α | $1 + 2\beta$ |
| Correlation | 0.911 | 0.882 | 0.693 | 0.785 |

Table 8.2: Different combinations of the coefficients α and β in OSS. The correlation is computed using the real human judgement from the Miller & Charles's experiment.

In Section 5.4.2, the downward transfer between two concepts was estimated by using the expected score of a concept, i.e.: $S(x) = 1/2$. To verify if this assumption holds in practice, OSS was tested with various other possibilities ranging from the minimum APS in the ontology to 1. Formally, it means that the experiment was reproduced with the following equation:

$$sim_{OSS}(x, z) = 1 - \frac{\log(1 + \delta\hat{\beta}(z, LCA(x, z))) - \log(\hat{\alpha}(x, LCA(x, z)))}{\max D}, \quad (8.1)$$

where the coefficient δ is not 2 anymore (under the assumption that $S(x) = 1/2$), but instead ranges from a value equals to the minimum APS in the ontology to 1.

| δ | 1 | $\frac{4}{3}$ | 2 | 4 | $\frac{1}{\min APS}$ |
|-------------|-------|---------------|-------|-------|----------------------|
| Correlation | 0.910 | 0.910 | 0.911 | 0.910 | 0.814 |

Table 8.3: Correlation with human judgement when coefficient δ ranges from a value equals to the minimum APS in the ontology to 1. Note that under the assumption that $S(x) = 1/2$, then δ is equal to 2.

Table 8.3 shows that the optimum value does in fact occur when $S(x) = 1/2$, but any value around that point will not greatly influence the correlation. However, big underestimations of the initial score tend to influence the correlation by over 10% as it will overestimate the coefficient β .

8.1.2 Experiment II - GeneOntology

To see whether previous results can be generalized, another experiment was performed on a much bigger scale using the GeneOntology (GO). GO was chosen over other ontologies as it is one of the most important ontologies within the bioinformatics community, and with over 20000 concepts, it is also one of the biggest.

As the name suggested, GeneOntology is an ontology describing gene products. Formally, the ontology is a DAG, where a concept represents a gene, and where an edge models *is-a* or *part-of* relationships. By definition, a gene product might be associated with or located in one or more cellular components; it is active in one or more biological processes, during which it performs one or more molecular functions. Thus, the DAG is further decomposed into three orthogonal sub-ontologies: *molecular function* (MF), *biological process* (BP), and *cellular component* (CC). More information about the GeneOntology can be found in Appendix D.2.

As for most real life applications, there is no human data of similarity over which OSS could be benchmarked. Instead, [Lord et al., 2003] proposed to use the Basic Local Alignment Search Tool (BLAST - [Altschul, 1990]) as it shows some correlations with concept similarity. BLAST compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches. In other words, BLAST finds regions of local similarity between sequences. Thus, this experiment will use the output of BLAST to estimate the similarity between pairs of proteins.

Formally, the experiment was conducted as follows. First, the February 2006 releases of the SWISS-PROT protein database² (SPD), GO³, and BLAST⁴ were downloaded. Then, all the concepts that were present in both GO and SPD were selected. To reduce the noise and errors, the proteins that were not annotated by a traceable authors were also removed (evidence code = *TAS*). For all remaining concepts, a BLAST search⁵ was performed in order to get a list of similar proteins. From this list, three proteins were randomly selected; respectively one at the beginning, the middle and one at the end. As with the WordNet experiment, the idea behind this selection is to cover high, intermediate, and low levels of

²<ftp://ftp.ebi.ac.uk/pub/databases/>

³<http://www.geneontology.org/GO.downloads.shtml#ont>

⁴<http://ncbi.nih.gov/BLAST/download.shtml>

⁵`blastall -p blastp -d swissprot -i in.txt -o out.txt -e 1000 -v 1000`

concept similarity. During a search, BLAST associates a *score* to each result that measures the similarity with the input protein. Therefore, it is this score (after normalization) that has been used as benchmark measure.

After the BLAST searches, the similarities between the concepts representing the input proteins and the resulting ones were measured using all of the metrics defined in Section 5.4.1. Finally, the deviation between the normalized BLAST score and the similarity values of all the concepts was measured using the *mean absolute error* measure (MAE, [Herlocker et al., 2004], Section 2.1.2). For each of GO's sub-ontologies, Table 8.4 shows the deviation values (MAE) for all the similarity metrics.

| | Edge | Leacock | Resnik | Lin | Jiang | OSS |
|---------|-------|---------|--------|-------|-------|--------------|
| MF | 0.450 | 0.234 | 0.224 | 0.223 | 0.200 | 0.185 |
| BP | 0.392 | 0.275 | 0.314 | 0.312 | 0.269 | 0.259 |
| CC | 0.351 | 0.303 | 0.286 | 0.292 | 0.343 | 0.260 |
| Average | 0.398 | 0.271 | 0.274 | 0.276 | 0.271 | 0.235 |

Table 8.4: MAE of various similarity metrics on the three sub-ontologies of the GeneOntology: molecular Function (MF), biological process (BP), and cellular component(CC).

The results are very interesting in two points. First, it shows that none of the existing techniques dominates another one. For example, Jiang's metric has a lower deviation on the MF ontology than Resnik's metric, but it is not true for the CC ontology. These results can be explained by the fact that the topology of the sub-ontologies differ widely. For example, BP has 10796 concepts and 85.3% of *is-a* relations, MF has 7923 concepts and 99.9% of *is-a* relations, while CC has only 1181 concepts and 59.8% of *is-a* relations.

Finally, it can be seen that the OSS similarity function has the lowest deviation, whatever the sub-ontology. This suggests that OSS is more robust than existing techniques, and that it is also significantly more accurate (with p-values < 0.01). We believe that this results are achieved because the algorithm used in BLAST shares some similarities with our a-priori score.

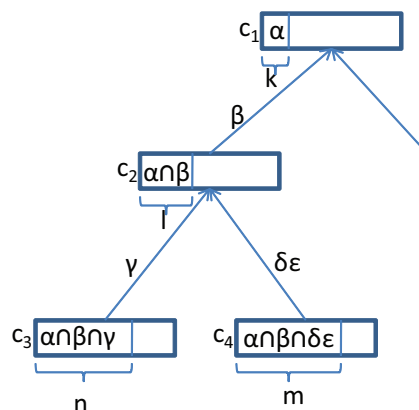


Figure 8.2: An ontology modeling the sequence describing genes, where the hierarchy is arranged based on what the genes have in common.

Imagine an ontology that represents genes' sequences, where the hierarchy is arranged in such way that concepts represent genes having the some part of their sequences that intersects. In such hierarchy, an edge represents the extra bit of sequence that differentiates a child concept from its parents. To simplify this example, we make the hypothesis that each concept has only one continuous subsequence in common with its parents, and that this common subsequence always starts from the beginning of the sequence. As a consequence, an ontology as shown in Figure 8.2 can be constructed. Note that such ontology can be easily constructed with real life sequences by breaking up the sequences into smaller part, and modeling each subpart with its own ontology. Multiple inheritance can then be used to link all the sub-parts together. The GeneOntology does share some commonalities with this model as concepts in the ontology represent families of genes that have common subsequences.

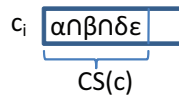


Figure 8.3: Illustration of common subsequence of a concept c , $CS(c)$.

Under these hypotheses, we define the common subsequence of a concept c , $CS(c)$, as a function that measures the length of the sequence that concept c and its descendants have in common. Moreover, we assume that the function is normalized in the interval $[0, 1]$. Figure 8.3 illustrates the CS function on concept c_i . Thus, the probability that the length of the subsequence of a concept c is superior to the threshold x , $P(CS(c) > x)$, is equal to $1 - x$. However, this probability ignores the fact that a concepts can have descendants, which directly influences this probability. Furthermore, for a concept c_k to be a parent of c_i , c_k must have part of its sequence that intersects with c_i . As a consequence, $CS(c_k) < CS(c_i)$ as concept c_i contains some subsequence that differentiates it from c_k . Therefore, the probability that the common length of any concept c is superior to a threshold x is equal to $(1 - x)^{n+1}$, where n is the number of descendants of c . Note that we count all descendants of c and c itself, to account for the fact that each concept may have instances that do not belong to any sub-concept. Thus, for a concept c to satisfy the function $CS(c)$, all the instances of its descendants must satisfy this function, as well as all the instances of c .

Thus, the probability distribution of the score for a concept c is $P(CS(c) \leq x) = 1 - (1 - x)^{n+1}$, with the following density function:

$$f_c(x) = \frac{d}{dx} (1 - (1 - x)^{n+1}) = (n + 1) \cdot (1 - x)^n \quad (8.2)$$

As a consequence, the expected lowest bound of the length of the common subsequence of a concept c , $E(CS(c))$, can be obtained by integration of the density function as follows.

$$\begin{aligned}
E(CS(c)) &= (n+1) \int_0^1 x(1-x)^n dx \\
&= (n+1) \left[\underbrace{\left. \frac{x(1-x)^{n+1}}{n+1} \right|_0^1}_0 + \underbrace{\int_0^1 \frac{(1-x)^{n+1}}{n+1}}_{1/((n+1)(n+2))} \right] \\
&= \frac{1}{n+2}
\end{aligned} \tag{8.3}$$

Equation 8.3 shows that the expected lowest bound of the length of the common subsequence concept c is in fact equal to the lowest bound of the score c , $APS(c)$.

8.2 Ontology filtering with a static ontology

The previous experiments have shown that OSS significantly outperforms state of the arts similarity metrics on WordNet and the GeneOntology. In short, these results showed that OSS is the most accurate at predicating human similarity judgement. Moreover, it tends to confirm the fact that the inference mechanism can successfully infer missing preferences, as OSS is derived from it.

Following these results, this section will look at whether or not the ontology filtering recommender system gives more accurate recommendations than existing recommender systems. Obviously, it is impossible to compare all the approaches, instead, this dissertation uses the item-based collaborative filtering as benchmark (see Section 2.4.2). This technique was used for two main reasons. First, CF has proven to be the most popular and most effective recommender system in practice. Second, ontology filtering uses the same data to learn the ontology as the collaborative filtering, which makes it a fair comparison. Note that the experiments that validate the learnt ontology approach will be described in the Section 8.3.

8.2.1 Experimental set-up

The data for this experiment is drawn from the famous MovieLens data set⁶. MovieLens is the most famous data set in the recommender system community; it contains the ratings of 943 real users on at least 20 movies. There are 1682 movies in total, which can be described by 19 themes: drama, action, and so forth. Using the weighted additive model, WADD, described in Section 2.3.3, a movie can be described as follows:

$$V(i) = \sum_{k=1}^p w_k v_k(i), \tag{8.4}$$

where v_k is the utility of the k^{th} theme, w_k is the weight of the k^{th} theme, and p is the total number of themes item i has. Thus, the theme of a movie can be seen as a discrete

⁶<http://www.cs.umn.edu/Research/GroupLens/data>

attribute taking the value 0 or 1, as a theme may or may not appear in a movie. As a consequence, the utility of a theme will be computed from the ratings a user has assigned to a movie. As a user expresses preferences on a small number of movies, only a few utility values can in fact be computed. To overcome this problem, the remaining utility values will be estimated using ontology filtering. Unfortunately, computing the weights is much harder, and would request asking the user too many elicitation questions. Instead, the equal weight policy was used [Zhang and Pu, 2005], as it has a very high relative accuracy compared to more complex ones. Formally, this policy assigns the same weight to each attribute, while making sure that the sum of these weights equals to 1. Note that more complex methods such as multiple regression can also be used to estimate such weights.

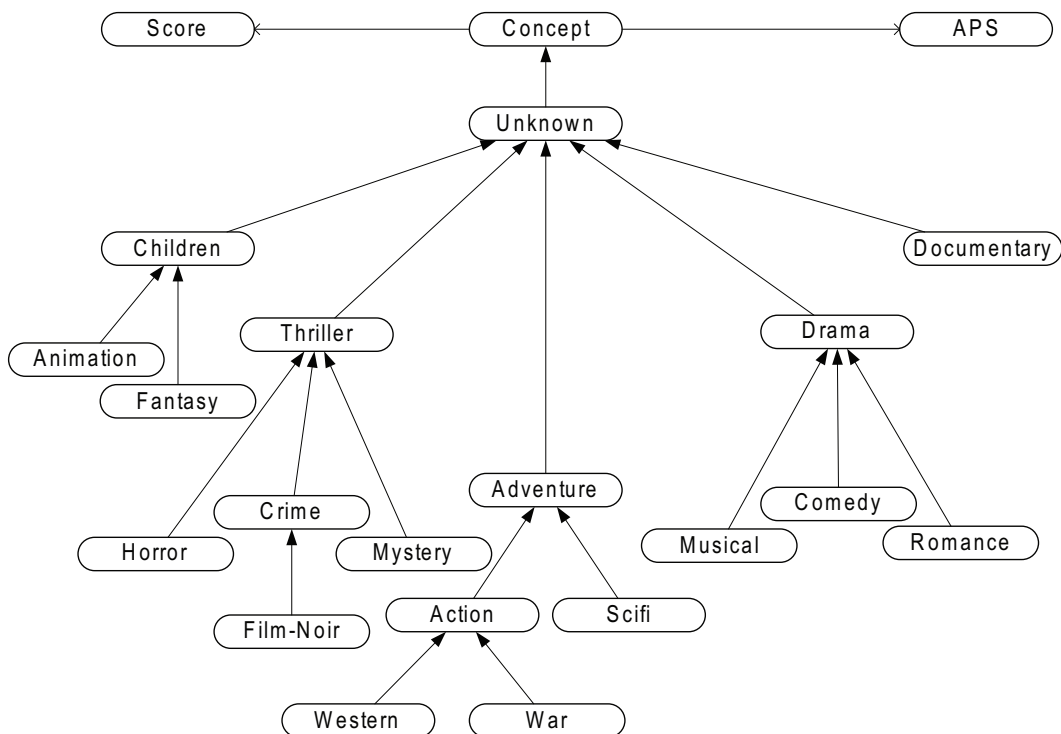


Figure 8.4: Hand made ontology for the MovieLens data set.

In Chapter 7.2.2, it is explained that an ontology can be created either from a consensus among a group of experts, or by using clustering algorithms. Using the movies' themes, it is in fact possible to design such ontology. With the help of some online dictionaries and IMDb.com⁷, the author of this thesis created the ontology illustrated in Figure 8.4. Note that the author of this thesis does not claim to be a movie expert, but created this ontology for the sake of this work. Many people would (and have) argued that this ontology is very basic, and could be refined by adding more information such as multiple hierarchy. However, as shown later in this section, experimental results show that ontology filtering with this simple ontology actually outperforms item-based collaborative filtering.

⁷<http://reviews.imdb.com/Sections/Genres/>

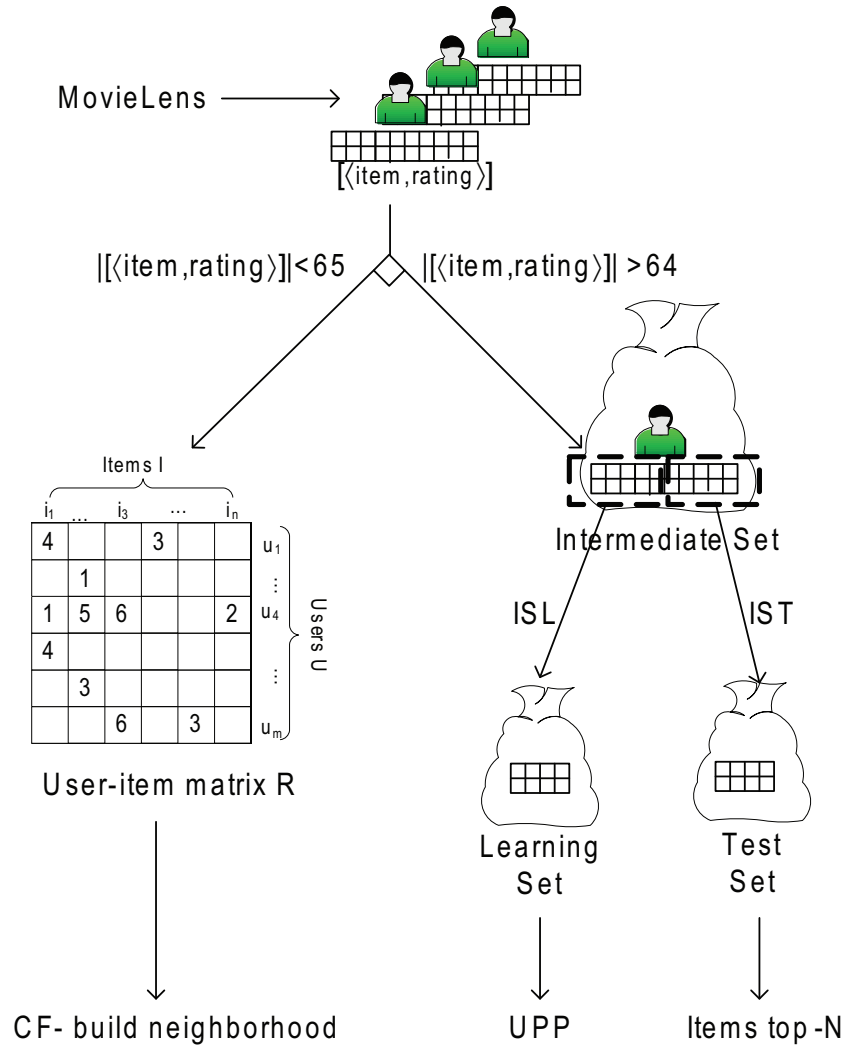


Figure 8.5: Illustration of the set up for experiments III and IV.

To evaluate ontology filtering, some training data must be extracted from the data set in order to build the users' preference profiles, and to test the recommendations. Formally, for all the experiments in this section, the training and test sets were obtained as follows (Figure 8.5). For each user in MovieLens, all of her ratings are inserted either in the *Intermediate set - IS* or in the *user-item matrix - R*. As its name indicates, the user-item matrix R contains all the users's preference profiles for the users who rated less than 65 movies. This matrix is used by collaborative filtering for building the neighborhood of items. On the other hand, the intermediate set IS is used for creating the users' preference profiles, and for testing the behavior of the recommender systems, where the set of users in IS is called U_{user} . For each remaining user in IS , its ratings are randomly split in two partitions: ISL and IST . The former set contains exactly 50 ratings, while the latter contains the remaining ones. These two intermediate sets are used to study the behavior of the recommendation algorithms with different amounts of rating data to learn the model. In the first scenario,

5 ratings are extracted from *ISL* and inserted into the user's *learning set* - *LS*, in order to simulate the case when few preferences about the user are known. Obviously, all the items in *IST* are inserted in the *test set* - *TS*, which is used for computing the prediction using the top-N recommendation policy, with N set to 5. Then, the experiment is reiterated by increasing the size of the learning set to respectively 10, 20, 40 and 50 rated items.

For each user in the learning set, the on-line phase of the ontology filtering process is applied in order to obtain some recommendations (see Section 7.2.2). To summarize this process, the 4 main steps are given below. For each user in the learning set, the user's preferences profile *UPP* is built by simply copying all the ratings from the user's learning set into the *UPP*. As the ontology filtering possesses just one ontology, Algorithm 8 simply returns this unique ontology. Given this ontology, Algorithm 5 builds the user's ontological profile by computing the score of the concept that has an instance in *UPP*. Then, missing scores are inferred using ontology filtering's inference mechanism. Finally, Algorithm 6 generates a set of 5 recommendations from the items found in the user's test set *TS*, and the mean absolute error is used to measure the accuracy of those recommendations. Note that this algorithm is slightly modified as an item can be instanced of one or more concepts. As a consequence, the overall score of an item is computed using the WADD model defined in Equation 8.4.

8.2.2 Experiment III - Behavior of ontology filtering

This experiment focuses on the ontology filtering approach, and looks at its behavior when modifying the way the weights and value function are computed. Thus, four distinct strategies have been implemented, which are as follows:

- *OF* is the ontology filtering process as defined in Section 7.2.2, where the missing utility values are inferred, and weights are computed using the equal weight policy.
- *Random inference* has the weights computed using the equal weight policy, but the missing utilities are replaced by random values.
- *Random weight* uses the inference mechanism to infer missing utility values, but uses random weights.
- *Random policy* uses random values for both the weights and utility values.

For each of this strategy, Algorithm 6 generates a set of 5 recommendations from the items found in the user's test set *TS*, where the hybrid score is computed only from the user's own score (i.e.: personalization coefficient ρ is set to 1).

Figure 8.6 shows the recommendation accuracy of the top-5 items using the mean absolute error metric (MAE, Section 2.1.2). As expected, it is the ontology filtering approach *OF* that performs the best, as it has the lowest mean absolute error. As the size of the user's learning set increases, more and more scores can be precisely learnt, which reduces the need of the inference mechanism. Note however that there is not a big variation in accuracy which tends to show that the inference mechanism is doing a good job. The *random inference* strategy always performs significantly worse than *OF* (p-value < 0.5), which

suggests that the inference mechanism can indeed estimate the user's preferences. Surprisingly, the ontology filtering approach produces recommendations which are much better than the *random weight* policy. This shows that the weights are also very important in the WADD model. On the other hand, the behavior of the random policy is not a surprise and is coherent with the fact that everything is randomly generated, which obviously gives worse predictions than with the user's preferences.

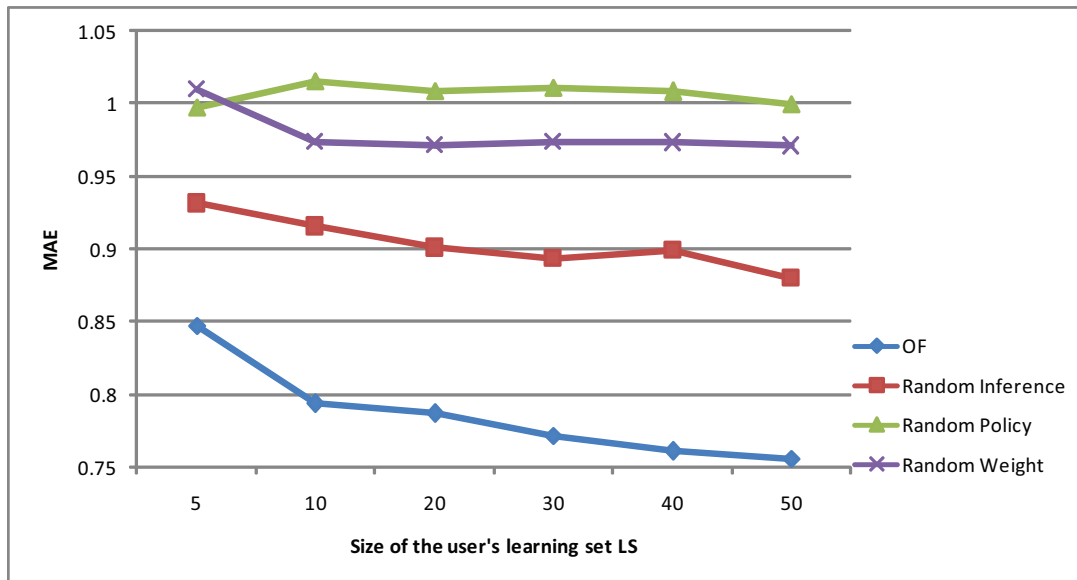


Figure 8.6: Ontology filtering versus other strategies for inferring the value functions and weights.

8.2.3 Experiment IV - Ontology filtering vs other techniques

This next experiment focuses on whether the ontology filtering recommender system with the learnt ontology is better or worse than existing approaches. Again, testing all the recommender systems is impossible. Instead, this experiment looked at the most popular approach on the web today - item-based collaborative filtering (CF). CF was also chosen because the ontology learning mechanism uses the same data. Furthermore, content approaches were not considered as they require a lot of information to compute an accurate model of the user, which is not available in this situation.

The set-up of the experiment remains the same as the previous experiment, but it is now the following 5 recommendation strategies which are being evaluated.

- *Random* is the most naive approach. It simply selects 5 items randomly from the user's test set TS .
- *Popularity* is a simple but very effective non-personalized strategy that ranks the movies based on their popularity. The popularity of each movie was computed using the user's rating in the user-item matrix R .

- *OF* is the ontology filtering process as defined in Section 7.2.2, where the missing utility values are inferred and weights are computed using the equal weight policy. However, the hybrid score of a concept is computed only from the user's score (i.e.: the personalization coefficient ρ is set to 1).
- *OF_pop* is the ontology filtering process as defined in Section 7.2.2, where the missing utility values are inferred and weights are computed using the equal weight policy. In this situation, the hybrid score of a concept is computed exactly as mentioned in Algorithm 6, which means that it combines both of the previous approach (i.e.: coefficient ρ is set to 0.5).
- *CF* is the item-based collaborative filtering, where the pairwise item similarities are computed using the adjusted cosine metric. The number of neighbors was set to 90 as [Mobasher et al., 2004] and [Sarwar et al., 2001] have shown that the optimum for MovieLens is very close to this value. Note that to reduce the bias, the author did not implement this algorithm himself, but instead used the freely available package Multilens⁸.

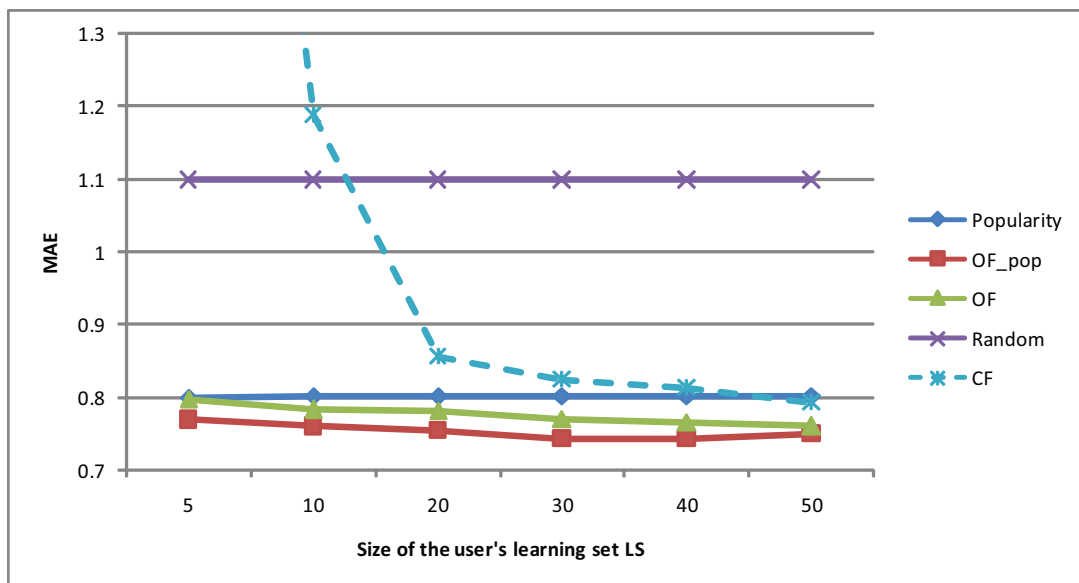


Figure 8.7: Ontology filtering versus other recommender strategies.

First, the predictive accuracy of each method is measured using various size of the learning set LS. Figure 8.7 plots the recommendation accuracy of the various recommender systems when different sizes of learning sets are used to build the users' preference profiles. This graph clearly shows the weakness of CF when only a few ratings are used to learn the model. This is known as the cold-start problem, where a minimum number of ratings need to be known in order to find the right neighborhood of similar users, i.e. at least 20 in this

⁸<http://knuth.luther.edu/~bmiller/multilens.html>

case. Note that when only 5 items are available, CF has a MAE of 2.59, which is twice as bad as the random policy. This simply reflects the fact that when so few ratings are available, then CF is totally unable to predict the ratings of some items. However, OF does not suffer from this problem and shows significant improvement over CF (p -value <0.01), when we have less than 30 ratings in the learning set. In more advanced experiments, CF was found to overcome OF when users had over 65 ratings in their preference profiles. Surprisingly, the popularity metric performs well, even better than CF when the number of ratings in $LS < 50$, which shows that users tend to like popular items. For example, take a random user who wants to watch a movie, and imagine that you have two choices; either the latest Harry Potter, or the Swiss movie Vitus. In this situation, most users will choose the Harry Potter movie as it is very famous, and most people heard about it. As expected, the best accuracy is obtained when OF is combined with the popularity approach. The combination of the scores allows the system to better discriminate between good and bad items with a higher confidence.

Second, the novelty of the recommendations generated with each approach was compared against the Popularity one. The motivation for this experiment lies in the previous movie example. Recommending a Harry Potter movie is actually not very *smart*, as most people may already have seen it or read the book. Instead, a smart recommender system should recommend items that a user will never have seen otherwise. Unfortunately, novelty is very hard to measure as it requires asking the user if he knew or heard about the items which have been predicted. Moreover, there is no standard way of measuring the novelty as there is for the accuracy. Instead, the dissertation used the novelty metric defined by Equation 2.4.

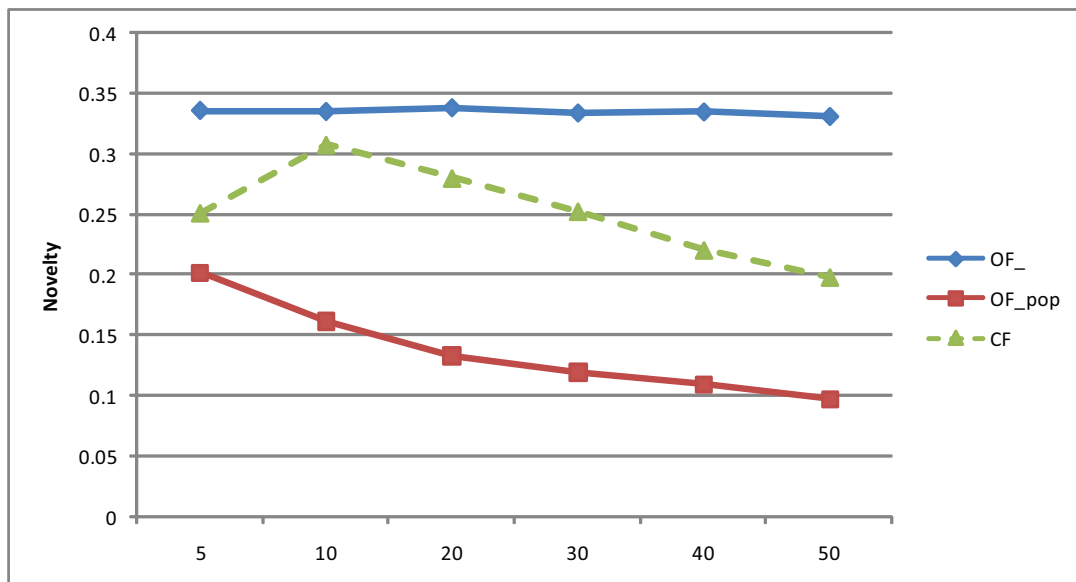


Figure 8.8: Novelty of various recommender techniques compared to the popularity approach.

Again, the results are very interesting in two points (Figure 8.8). First, it shows that it is OF that produces the best non-obvious recommendations, whatever the size of the learning set. The novelty value of ontology filtering is always greater than 33%, which means that a third of the recommendations were novel. Second, CF's novelty seems to improve when there are less than 10 ratings, and then decreases steadily down to the 20% threshold. This behavior can be explained if we superpose this result with the MAE. When there are less than 20 ratings, CF's accuracy is very low, which tends to indicate that items were selected from many diverse neighborhoods.

Finally, the ontology filtering with the hybrid score tells us that the use of popularity data can improve the overall recommendation accuracy over our simple OF approach, but this gain is then lost in recommendation novelty. Thus, a tradeoff must be done between the recommendation accuracy and its novelty. This also explains why the personalization coefficient ρ is set to 0.5, half way between pure personalization and pure popularity recommendation.

8.3 Ontology filtering with the learnt taxonomies

The previous two experiments have shown that ontology filtering with a predefined ontology can successfully estimate missing preferences. Furthermore, they have shown that ontology filtering can outperform item-based collaborative filtering, while keeping a higher degree of novelty in the recommendation. Finally, it was shown that the popularity of items can be used to increase the prediction accuracy (Algorithm 6), but at a cost in novelty.

Following these results, the rest of this chapter looks at whether or not the ontology filtering recommender system with the learnt taxonomies can also give more accurate recommendations than item-based collaborative filtering. The taxonomies generated by the various clustering algorithms are also studied, and results show that no unique clustering algorithm is able to rule the other ones. Moreover, it is shown that hierarchical distance-based clustering algorithms are sensitive to both the size of the user's preferences and the size of the clustering trees. Fortunately, experimental results show that personalizing the ontology based on the user preferences does help to increase the recommendation accuracy, while reducing the problems faced by single clustering algorithms. At the same time, the intuition that letting some concepts have more than one parents is verified, but experimental results on Jester shows that it is at a cost in computational complexity.

Note that Appendix E.2 contains an experiment that compares ontology filtering with the various ontologies. Unfortunately, due to the bias in the nature of the experience set-up, it has not been included in this chapter but in Appendix E for further information.

8.3.1 Experimental set-up

To evaluate ontology filtering with the learnt taxonomies, several experiments have been performed on two famous data sets: MovieLens⁹ and Jester¹⁰. MovieLens is the data set used in Experiment III and IV, while Jester is another famous data set that contains the

⁹<http://www.cs.umn.edu/Research/GroupLens/data>

¹⁰<http://www.ieor.berkeley.edu/~goldberg/jester-data/>

users' ratings on jokes collected over a period of 4 years. The data set contains over 4.1 Million ratings and is actually split into three zip files: jester-data-1.zip, jester-data-2.zip, jester-data-3.zip. In this dissertation, only the first data set was used as it contains already 24,983 users on all the 100 jokes.

These sets were used for three reasons. First and most importantly, they are the most widely used data sets, which makes it easy for other researchers to reproduce the experiments. Second, both of those data sets contain (real) human ratings on items, which are necessary for filling in the user-item matrix R , and learning the ontologies. Finally, these sets are very different in content and sparsity, where the sparsity is defined as the fraction of entries in the matrix R without values [Sarwar et al., 2001] over the total number of possible entries. For example, MovieLens contains movies that can easily be defined with some features such as the theme, duration, and so forth. Jester is made up of jokes, which is much harder to describe, and thus, is an ideal candidate to test our ontology learning algorithm. Notice also that the sparsity of MovieLens(0.937) is much greater than the one of Jester(0.447).

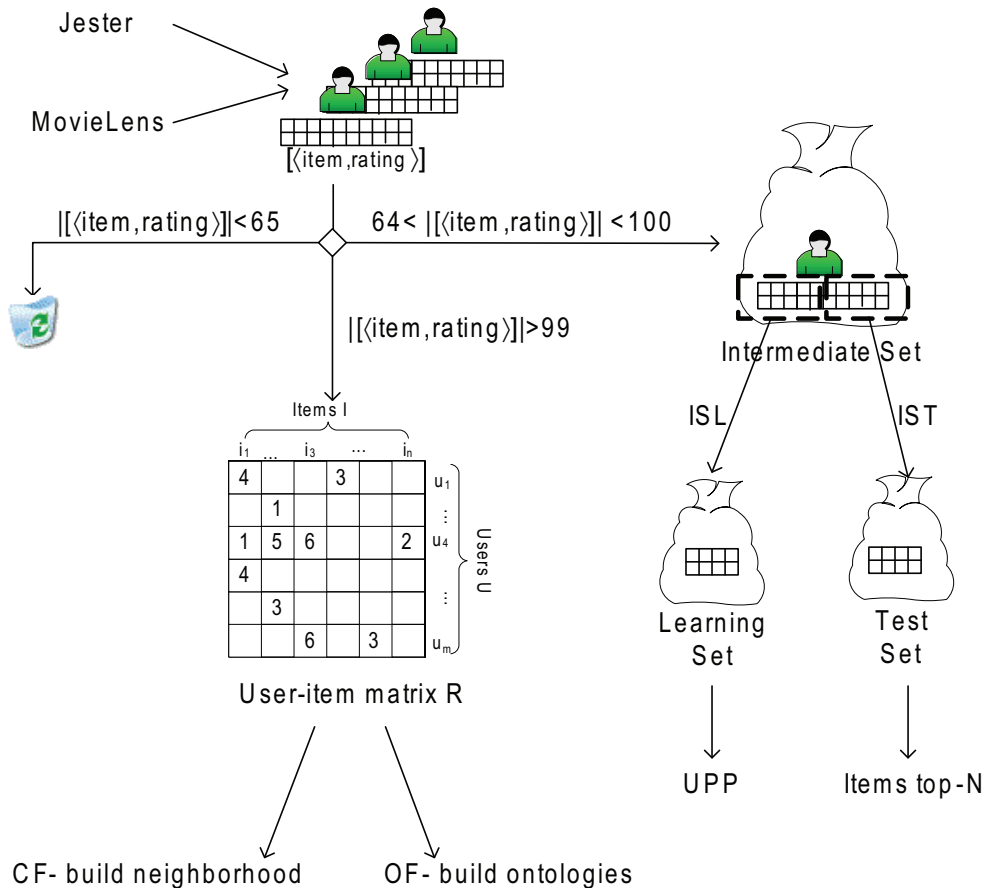


Figure 8.9: Illustration of the set up for experiments V to VIII.

As with the previous experiments, some training data must be extracted from these data sets in order to build the preference profiles, and the ontologies. Formally, for all

the experiments in this section, the training and test sets were obtained as follows (Figure 8.9). First, all the users who rated less than 65 items were removed. The remaining users were then split in two distinct partitions: *Intermediate set - IS* and *user-item matrix - R*. As its name indicates, the user-item matrix R contains all the users' preference profiles for the users who rated at least 100 items. This matrix is used by collaborative filtering for building the neighborhood, while the same data is used by ontology filtering for learning the ontology. On the other hand, the intermediate set IS is used for creating the user's preference profile and testing the behavior of the recommender systems. Thus, for each remaining user in IS , its ratings are randomly split in two partitions: ISL and IST . The former set contains exactly 50 ratings, while the latter contains the remaining ones. Note that in the previous experiments, the collaborative filtering model was learnt on users who rated less than 65 items. This set-up forces the CF's model and the ontologies to be learnt on users who rated at least 100 items. There are two reasons for this. First, CF and clustering algorithms perform better when the matrix R is not too sparse. Second, it is possible to test both CF and OF in various sparsity situations, as the Jester data set contains 100 jokes and some users have rated all of them.

In the first scenario, 5 ratings are extracted from ISL and inserted into the user's *learning set - LS*, in order to simulate the case when few preferences about the user are known. Obviously, all the items in IST are inserted in the *test set - TS*, which is used for computing the prediction using the top-N recommendation policy, with N set to 5. In the second scenario, all 50 items in ISL are inserted in LS to see how CF and OF behave when sufficient ratings are available for learning the model. Note that the experiments will be first executed with the first scenario, and then with the second. Note also that the first scenario is identified on the graphs with the notation $* - 5LS$, where $*$ is either OF or CF ; while the second one used the notation $* - 50LS$.

Collaborative filtering requires the number of neighbors k to be set before making the recommendation. Similarly, ontology filtering used a parameter θ that sets the number of leaf clusters in the ontology. To reduce the bias, the experiments will always use the same value for both k and θ . The recommendation accuracy of the predictions made by both CF and OF are measured using the standard F1 metric (Section 2.1).

For the clustering algorithms, the toolkit CLUTO¹¹ version 2.1.1 was used as it implements all of the partitional and agglomerative clustering algorithms used by Algorithm 7. On the other hand, the java package WEKA¹² version 3.4 was used for the COBWEB algorithm.

8.3.2 Experiment V - Evaluating the item-to-item similarity metrics

Ontology filtering uses ontologies to infer missing preferences. In Section 6.3, Algorithms 7 and 9 were defined to learn the underlying structures of these ontologies, followed by Algorithm 8 for selecting which one to use. The clustering algorithms used in Algorithms 7 and 9 rely on the item-to-item similarity for evaluating the distances between each

¹¹<http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download>

¹²<http://www.cs.waikato.ac.nz/~ml/weka/>

item. It was suggested in these algorithms to use adjusted cosine for computing all the pairwise similarities from the user-item matrix R .

In the literature, it is argued that the adjusted cosine is the best performing metric when performing item-based collaborative filtering, (CF). This is because the adjusted cosine can take into account the difference in rating scale between two users. To test whether the similarity metric influences the ontology generated using Algorithm 7, the following experiment was preformed. Using Pearson correlation - Equation 2.19, cosine similarity - Equation 2.20, and the adjusted cosine similarity - Equation 2.24 iteratively, all the pairwise similarities were generated from the user-item matrix R , which were then used in Algorithm 7 to learn a set of taxonomies. For each user in U_{user} , the best ontology was selected using Algorithm 8. This ontology is then used to infer the missing user's preferences, and for predicting the ratings of the items in the user's test set TS . This experiment is reiterated using various values for the threshold θ in Algorithm 7, with respectively 5 and 50 items in the user's learning set LS . The same experiment was also performed in parallel with the item-based collaborative filtering, where the number of neighbors k is always equal to θ .

| | Collaborative Filtering | | | Ontology Filtering | | |
|-----------|-------------------------|--------|-----------------|--------------------|--------|-----------------|
| | Pearson | Cosine | Adjusted Cosine | Pearson | Cosine | Adjusted Cosine |
| Jester | 0.324 | 0.310 | 0.328 | 0.364 | 0.364 | 0.368 |
| MovieLens | 0.176 | 0.161 | 0.176 | 0.278 | 0.286 | 0.290 |
| Average | 0.250 | 0.236 | 0.252 | 0.321 | 0.325 | 0.329 |
| STDEV | 0.009 | | | 0.004 | | |

Table 8.5: Average recommendation accuracy of CF and OF when using different similarity metrics to build the item-to-item similarity matrix S . The accuracy is measured by averaging the F1 value of all the users.

Table 8.5 shows the average prediction accuracies of the recommendations made by ontology filtering algorithms and item-based collaborative filtering. From these results, two very interesting observations arise. First, with an average F1 value of 0.329, the best predication accuracy is obtained when ontology filtering uses the adjusted cosine similarity metric to compute the item-to-item similarities. However, the improvement in accuracy is not statistically significant over the traditional cosine and Pearson measures. This tends to show that the similarity metric on the generated ontology is the most important factor.

As expected, it is the adjusted cosine similarity that gives the best decision accuracy for collaborative filtering. This confirms previously obtained results in the literature. However, theses results show that the similarity metric has a greater influence on collaborative filtering. The standard deviation of the three different metrics for CF is equal to 0.009, which is more than double the one of ontology filtering.

8.3.3 Experiment VI - Evaluating Algorithms 7 and 8

In this next experiment, the performance of the taxonomies generated by Algorithm 7 is studied, and whether or not the ontology personalization made by Algorithm 8 helps to

increase the prediction accuracy. Using Table 6.2, Algorithm 7 generates 15 different ontologies: 6 using the partitional approach, and 9 with the agglomerative approach. Table 8.6 summarizes the various algorithms and notations used in this experiment. Note that the COBWEB algorithm is used to benchmark the learnt ontologies. For all the users in U_{user} , the 15 ontologies were then used individually to create the user’s ontological profile, and infer the missing preferences of all the concepts. Then, the top-N recommendations were generated using the inferred ontological profile over the item in the user’s test set TS , where the hybrid score of a concept is set to the score of that concept (i.e.: the popularity of a concept is not considered $\Rightarrow \rho = 1$).

| | |
|--------------------|---|
| P- \mathcal{I}_1 | Partitional clustering using the criterion function \mathcal{I}_1 |
| P- \mathcal{I}_2 | Partitional clustering using the criterion function \mathcal{I}_2 |
| P- \mathcal{E}_1 | Partitional clustering using the criterion function \mathcal{E}_1 |
| P- \mathcal{G}_1 | Partitional clustering using the criterion function \mathcal{G}_1 |
| P- \mathcal{H}_1 | Partitional clustering using the criterion function \mathcal{H}_1 |
| P- \mathcal{H}_2 | Partitional clustering using the criterion function \mathcal{H}_2 |
| A- \mathcal{I}_1 | Agglomerative clustering using the criterion function \mathcal{I}_1 |
| A- \mathcal{I}_2 | Agglomerative clustering using the criterion function \mathcal{I}_2 |
| A- \mathcal{E}_1 | Agglomerative clustering using the criterion function \mathcal{E}_1 |
| A- \mathcal{G}_1 | Agglomerative clustering using the criterion function \mathcal{G}_1 |
| A- \mathcal{H}_1 | Agglomerative clustering using the criterion function \mathcal{H}_1 |
| A- \mathcal{H}_2 | Agglomerative clustering using the criterion function \mathcal{H}_2 |
| A- \mathcal{UP} | Agglomerative clustering using the criterion function UPGMA |
| A- \mathcal{SL} | Agglomerative clustering using the criterion <i>slink</i> |
| A- \mathcal{CL} | Agglomerative clustering using the criterion <i>clink</i> |
| COB | The incremental concept-based clustering algorithm COBWEB |
| OF | Ontology filtering using Algorithm 7 and 8 |

Table 8.6: Notations used by the various algorithms in Experiment VI.

As for the previous experiment, the prediction accuracy is measured using the F1 metric defined in Equation 2.3. The underlying assumption is that a good ontology should generate good recommendations, and thus increase the prediction accuracy. However, the results obtained by the different criteria functions significantly diverge within the same family of algorithms (i.e.: partitional or agglomerative). As a consequence, using a simple average measure over all the criteria would induce too much bias, especially when we consider the fact that there are more agglomerative algorithms than partitional ones. Thus, the F1 results were transformed into *relative F1 value*, $rF1$. This is done by dividing the F1 results by the maximum possible value between the 15 possible ontologies. Notice that the relative F1 value is very similar to the relative Fscore introduced in [Zhao and Karypis, 2005]. To have a better understanding of the behavior of each ontology, they have been tested with different numbers of leaf clusters, ranging from 5 to the number of items available.

Table 8.7 displays the relative F1 values, where each result element of the table is composed of two sub-elements x, y . The first element x was obtained when only 5 items were present in the user’s learning set LS , while y was obtained when the user had 50 items

in its learning set. The first 15 rows of Table 8.7 contain the accuracy of the recommendations obtained using the ontology learnt by a particular clustering algorithm. The row *COB* contains the recommendation accuracy when only the COBWEB algorithm is used to learn the ontology, while the last row shows the full ontology filtering process as defined in Algorithms 7, and 8. In short, this experiment observed the behavior of a combination of ontologies, when users had respectively very small and moderate preference profile.

| | 5 | 10 | 20 | 40 | 60 | 80 | 100 | AVG |
|--------------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-------------------|
| P- \mathcal{I}_1 | 0.92,0.92 | 0.97,1.00 | 1.00,0.95 | 0.98,0.92 | 0.93,0.95 | 0.89,0.90 | 0.90,0.94 | 0.94,0.94 |
| P- \mathcal{I}_2 | 1.00,1.00 | 0.99,0.94 | 0.98,0.95 | 0.98,0.95 | 0.90,0.92 | 0.90,0.91 | 0.90,0.91 | 0.94,0.94 |
| P- \mathcal{E}_1 | 0.72,0.86 | 0.69,0.77 | 0.78,0.70 | 0.79,0.82 | 0.76,0.71 | 0.76,0.73 | 0.85,0.90 | 0.77,0.78 |
| P- \mathcal{G}_1 | 0.67,0.60 | 0.72,0.62 | 0.74,0.68 | 0.91,0.82 | 0.98,0.97 | 1.00,1.00 | 1.00,0.89 | 0.86,0.80 |
| P- \mathcal{H}_1 | 0.96,0.99 | 1.00,1.00 | 0.93,1.00 | 0.89,1.00 | 0.91,0.95 | 0.87,0.87 | 0.85,0.90 | 0.92, 0.96 |
| P- \mathcal{H}_2 | 0.96,0.94 | 0.98,0.98 | 0.94,0.97 | 0.87,0.95 | 0.78,0.90 | 0.81,0.85 | 0.85,0.90 | 0.89,0.93 |
| A- \mathcal{I}_1 | 0.92,0.98 | 0.95,0.97 | 0.97,0.92 | 0.96,0.92 | 0.93,0.95 | 0.89,0.90 | 0.90,0.94 | 0.93,0.94 |
| A- \mathcal{I}_2 | 0.94,0.96 | 0.97,0.97 | 0.98,0.93 | 0.93,0.91 | 0.89,0.91 | 0.84,0.92 | 0.88,0.96 | 0.92,0.94 |
| A- \mathcal{E}_1 | 0.79,0.91 | 0.73,0.84 | 0.67,0.61 | 0.67,0.60 | 0.71,0.65 | 0.80,0.76 | 0.85,0.91 | 0.75,0.75 |
| A- \mathcal{G}_1 | 0.67,0.60 | 0.72,0.63 | 0.77,0.69 | 0.91,0.82 | 1.00,0.96 | 1.00,0.96 | 0.97,0.94 | 0.86,0.80 |
| A- \mathcal{H}_1 | 0.74,0.73 | 0.83,0.77 | 0.77,0.72 | 0.80,0.74 | 0.83,0.80 | 0.84,0.82 | 0.86,0.93 | 0.81,0.78 |
| A- \mathcal{H}_2 | 0.85,0.84 | 0.83,0.85 | 0.81,0.76 | 0.73,0.68 | 0.85,0.81 | 0.88,0.86 | 0.90,1.00 | 0.84,0.83 |
| A- \mathcal{UP} | 0.94,0.95 | 0.98,0.93 | 0.96,0.92 | 0.97,0.96 | 0.94,0.96 | 0.90,0.91 | 0.88,0.98 | 0.94,0.94 |
| A- \mathcal{SL} | 0.67,0.60 | 0.69,0.60 | 0.70,0.64 | 1.00,0.97 | 1.00,1.00 | 0.96,0.92 | 0.97,0.98 | 0.85,0.82 |
| A- \mathcal{CL} | 0.95,1.00 | 0.96,0.97 | 0.96,0.96 | 0.97,0.92 | 0.95,0.95 | 0.89,0.90 | 0.89,0.97 | 0.94,0.95 |
| COB | 0.70,0.65 | 0.70,0.64 | 0.67,0.64 | 0.64,0.63 | 0.62,0.63 | 0.61,0.62 | 0.63,0.70 | 0.65,0.64 |
| OF | 1.07,1.11 | 1.10,1.08 | 1.08,1.05 | 1.08,1.05 | 1.05,1.06 | 0.94,1.02 | 0.92,0.96 | 1.04,1.05 |

Table 8.7: Relative F1 values obtained on the Jester dataset. The notation x, y means that users in U_{user} had respectively 5 and 50 ratings in their learning set LS . The number of leaf clusters ranges from 5 to 100.

From this table, it can be seen that, on average, ontology filtering using Algorithm 8 to select the best learnt ontology performs better than any of the simple clustering algorithms alone ($rF1 = 1.04$ and $rF1 = 1.05$), whatever the size of the users' learning set. This tends to confirm our intuition that not one ontology is strictly better than the others, but rather that some users will reach a better accuracy with a given ontology. Notice that having a relative score superior to 1 for OF is not a mistake. It is due to the fact that the maximum value for the normalization is selected from the 15 simple clustering ontologies only. Another important result is that the ontology learnt by COBWEB is the worst performing ontology. The main reason for this lies in the definition of COBWEB: it is a *conceptual* clustering algorithm. As a consequence, items need to be defined by a set of attribute-value pairs. In our eCommerce context, this data is unavailable as we only have a set of rated items. This is also an indication that the technique [Clerkin et al., 2001] of transforming the item-to-item similarity matrix into attribute-value pairs is not suitable in this domain. When 5 items are used for learning the user's score, the partitional clustering with the \mathcal{I}_2 criteria function (P- \mathcal{I}_2) has the best average relative score. Then, partitional clustering with the \mathcal{H}_1 function (P- \mathcal{H}_1) becomes the best algorithm when 50 items are used for building the user's preference profile. This tends to go in the direction of Zhao's conclusions [Zhao and Karypis, 2005] that partitional clustering algorithms perform better than agglomerative ones. Notice the

evolution of the clustering $P\mathcal{G}_1$ with the number of clusters. When few leaf clusters are created, then $P\mathcal{G}_1$ performs poorly. However, when we have many clusters, $P\mathcal{G}_1$ performs really well, which tends to show that the graph criterion requires many leaf clusters to generate a good ontology. When the number of clusters are set to either 40, 60, or 80, and LS to 5, then interestingly agglomerative algorithms do perform better than partitional clustering ones. This would indicate (and the results with MovieLens confirm this) that agglomerative clustering needs enough leaf clusters to perform correctly, while partitional algorithm seems better with less clusters. It makes sense when we consider the fact that partitional algorithms are top down approaches that recursively split clusters. Thus, too many splits may degrade the ontology, as it generally increases the variance. Inversely, agglomerative clusterings are bottom up approaches that recursively merge clusters.

| | 5 | 20 | 60 | 100 | 500 | 1000 | 1668 | Average |
|------------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-------------------|
| $P\mathcal{I}_1$ | 0.84,0.86 | 0.92,0.91 | 0.99,0.93 | 0.94,0.90 | 1.00,0.87 | 1.00,0.88 | 0.99,0.86 | 0.95,0.89 |
| $P\mathcal{I}_2$ | 0.88,0.78 | 0.80,0.93 | 0.80,0.97 | 0.74,0.89 | 0.81,0.87 | 0.97,0.87 | 0.87,0.87 | 0.84,0.88 |
| $P\mathcal{E}_1$ | 0.90,1.00 | 0.84,0.94 | 0.69,0.92 | 0.64,0.89 | 0.68,0.72 | 0.64,0.68 | 0.89,0.90 | 0.75,0.83 |
| $P\mathcal{G}_1$ | 0.92,0.87 | 0.88,0.88 | 0.72,0.92 | 0.68,0.90 | 0.82,0.89 | 0.88,0.79 | 0.88,0.79 | 0.83,0.86 |
| $P\mathcal{H}_1$ | 0.93,0.94 | 1.00,1.00 | 0.80,1.00 | 0.74,0.93 | 0.99,0.96 | 0.92,0.91 | 0.82,0.91 | 0.89, 0.95 |
| $P\mathcal{H}_2$ | 0.88,0.92 | 0.82,0.85 | 0.67,0.89 | 0.63,0.86 | 0.70,0.84 | 0.70,0.72 | 0.80,0.73 | 0.74,0.83 |
| $A\mathcal{I}_1$ | 0.73,0.52 | 0.65,0.51 | 0.52,0.54 | 0.48,0.50 | 0.51,0.48 | 0.91,0.81 | 1.00,0.81 | 0.69,0.59 |
| $A\mathcal{I}_2$ | 0.85,0.65 | 0.75,0.59 | 0.61,0.62 | 0.57,0.60 | 0.58,0.60 | 0.90,0.82 | 0.91,0.80 | 0.74,0.67 |
| $A\mathcal{E}_1$ | 0.75,0.59 | 0.65,0.53 | 0.53,0.55 | 0.48,0.52 | 0.75,0.85 | 0.85,1.00 | 0.81,1.00 | 0.69,0.72 |
| $A\mathcal{G}_1$ | 0.75,0.57 | 0.67,0.58 | 0.55,0.67 | 0.54,0.71 | 0.67,0.74 | 0.91,0.80 | 0.82,0.79 | 0.70,0.69 |
| $A\mathcal{H}_1$ | 0.88,0.83 | 0.91,0.90 | 0.75,0.91 | 0.79,0.92 | 0.86,0.91 | 0.84,0.79 | 0.70,0.79 | 0.82,0.86 |
| $A\mathcal{H}_2$ | 1.00,0.90 | 0.88,0.90 | 0.77,0.91 | 0.84,0.92 | 0.87,0.86 | 0.83,0.77 | 0.74,0.75 | 0.85,0.86 |
| $A\mathcal{UP}$ | 0.88,0.72 | 0.77,0.66 | 0.62,0.69 | 0.58,0.66 | 0.61,0.64 | 0.92,0.85 | 0.96,0.83 | 0.76,0.72 |
| $A\mathcal{SL}$ | 0.73,0.52 | 0.64,0.48 | 0.53,0.56 | 0.48,0.53 | 0.51,0.48 | 0.61,0.70 | 0.90,0.68 | 0.63,0.56 |
| $A\mathcal{CL}$ | 0.89,0.91 | 0.92,0.92 | 1.00,0.93 | 1.00,1.00 | 0.98,1.00 | 0.94,0.79 | 0.97,0.78 | 0.96,0.90 |
| COB | 0.83,0.71 | 0.72,0.64 | 0.59,0.67 | 0.55,0.65 | 0.58,0.64 | 0.52,0.55 | 0.51,0.92 | 0.62,0.68 |
| OF | 1.21,1.27 | 1.20,1.34 | 1.18,1.34 | 1.10,1.35 | 1.21,1.28 | 1.08,1.20 | 0.81,1.07 | 1.11,1.26 |

Table 8.8: Relative F1 values obtained on the MovieLens dataset. The notation x, y means that users in U_{user} had respectively 5 and 50 ratings in their learning set LS . The number of leaf clusters ranges from 5 to 1668.

As shown in Table 8.8, the MovieLens results are very similar to those obtained with the Jester data set. For example, personalized ontology filtering performs better on average than any of the clustering algorithms taken separately, whatever the size of the learning set LS . As a matter of fact, the improvement is even more significant than with Jester. Second, the ontology produced by COBWEB is again giving a very poor prediction accuracy. For the first time, the agglomerative clustering with the clink criteria function is the best performing clustering algorithm when 5 items are used for learning the user’s model. Note however, that $P\mathcal{I}_1$ is nearly as good as $A\mathcal{CL}$, and thus no clear conclusion can be drawn from this result. When 50 ratings are used to populate the user’s learning set LS , the best performing ontology is again the one learnt using the partitional clustering with the \mathcal{H}_1 function. This tends to suggest that, on average and when sufficient data about the user is known, partitional clustering with the \mathcal{H}_1 criteria function performs the best. A detailed analysis of the ontology size reveals in fact that the agglomerative clustering with the *clink*

function can outperform $P\text{-}\mathcal{I}_1$ if the number of clusters are set to either 100 or 500. Fortunately, our ontology filtering approach with Algorithm 8 is capable of predicting which ontology to use as its relative F1 score is always bigger than 1, and whenever the number of clusters are less or equal to 1000.

| | Jester | MovieLens | Average |
|--------------------------|-----------|-----------------|-------------------|
| $P\text{-}\mathcal{I}_1$ | 0.25,1.27 | 147.42,4.39 | 73.83,2.83 |
| $P\text{-}\mathcal{I}_2$ | 0.26,1.35 | 185.81,5.53 | 93.04,3.44 |
| $P\text{-}\mathcal{E}_1$ | 0.27,1.39 | 196.62,5.86 | 98.45,3.62 |
| $P\text{-}\mathcal{G}_1$ | 0.44,2.22 | 120.77,2.22 | 60.60,2.91 |
| $P\text{-}\mathcal{H}_1$ | 0.47,2.39 | 1293.98,38.53 | 647.22,20.46 |
| $P\text{-}\mathcal{H}_2$ | 0.53,2.71 | 1592.28,47.42 | 796.41,25.06 |
| $A\text{-}\mathcal{I}_1$ | 0.20,1.01 | 34.79,1.04 | 17.49,1.02 |
| $A\text{-}\mathcal{I}_2$ | 0.21,1.07 | 35.90,1.07 | 18.05,1.08 |
| $A\text{-}\mathcal{E}_1$ | 0.22,1.10 | 37.60,1.12 | 18.91,1.11 |
| $A\text{-}\mathcal{G}_1$ | 0.22,1.12 | 33.98,1.01 | 17.10,1.07 |
| $A\text{-}\mathcal{H}_1$ | 0.33,1.70 | 694.59,20.68 | 347.46,11.19 |
| $A\text{-}\mathcal{H}_2$ | 0.36,1.84 | 835.54,24.88 | 417.95,13.36 |
| $A\text{-}\mathcal{UP}$ | 0.20,1.01 | 36.59,1.09 | 18.39,1.05 |
| $A\text{-}\mathcal{SL}$ | 0.20,1.00 | 37.23,1.11 | 18.71,1.05 |
| $A\text{-}\mathcal{CL}$ | 0.20,1.00 | 33.58,1.00 | 16.89,1.00 |
| COB | 0.75,3.84 | 30816.00,917.67 | 15408.38,460.75 |

Table 8.9: Execution time T required for the clustering algorithm to generate the ontologies (in seconds). The element x of the tuple x, y is in seconds, while y measures the relative time rT from the best clustering algorithm (1.0 being the best).

When considering which clustering algorithm to use, the accuracy of the prediction should not be the only criterion. The execution time required to build the ontology should also be an important aspect. Table 8.9 displays the execution time (in seconds) required for the clustering algorithm to generate its tree, and the relative execution time. The relative execution time of an algorithm a , $rT(a)$, is computed as the ratio of the execution time T of algorithm a over the minimum time T of any of the 15 algorithms. As expected, the execution time does vary a lot from one algorithm to another. The worst performing algorithm is COBWEB, which requires nearly 9 hours to come up with a clustering tree on MovieLens! This result was expected as the complexity is exponential to the number of attribute-value pairs, which in this case is equal to the 1682 different items (attributes) and two values (good or bad). As a consequence, COBWEB clearly does not scale well to big problems. However, it is good to point out that this computation can be broken into smaller parts, as COBWEB is an incremental algorithm. Very surprisingly, agglomerative clusterings took less time to compute than partitional clusterings, which is obviously not what is expected when reading the general literature. This is not a mistake, and there are two explanations for this. First, the most expensive step in agglomerative clustering is the pairwise similarity computation of all the pairs of items. In ontology filtering, the matrix S containing this data has already been computed, and thus removes this $\mathcal{O}(n^2)$ step. Second, as highlighted by Zhao [Zhao and Karypis, 2005], the pair-wise similarities of the improvements in the value of the criterion function achieved by merging a pair of clusters

i and j do not change during the different agglomerative steps, as long as i and j are not selected to be merged. Finally, Table 8.9 shows that using the criteria functions \mathcal{H}_1 and \mathcal{H}_2 significantly increases the execution time of the clustering algorithm. These results are coherent with the theory as \mathcal{H}_1 and \mathcal{H}_2 are both hybrid functions that respectively combine criteria \mathcal{I}_1 with \mathcal{E}_1 , and \mathcal{I}_2 with \mathcal{E}_1 .

Previous results suggested that a compromise between the predication accuracy and the clustering time has to be done. For example, when users have 50 items in their training set, it is the partitional clustering algorithm with the \mathcal{H}_1 criterion function that performs the best. However, it takes over 1200 seconds for P- \mathcal{H}_2 to generate the tree, which is 20 times more than the fastest algorithm. As a consequence, this dissertation proposes to combine the execution time together with the prediction accuracy in order to measure the global efficiency of a clustering algorithm. Thus, the *efficiency* of a clustering algorithm a , $Eff(a)$, is defined as follows:

$$Eff(a) = \frac{rF1(a)}{rT(a)}, \quad (8.5)$$

where $rF1(a)$ is the relative F1 of algorithm a , and $rT(a)$ is the relative time required by algorithm a to build the taxonomy.

| | Jester | MovieLens | Average | |
|--------------------|--------|-----------|--------------|------|
| P- \mathcal{I}_1 | 0.744 | 0.209 | 0.477 | 0.32 |
| P- \mathcal{I}_2 | 0.697 | 0.156 | 0.427 | |
| P- \mathcal{E}_1 | 0.558 | 0.136 | 0.347 | |
| P- \mathcal{G}_1 | 0.373 | 0.235 | 0.304 | |
| P- \mathcal{H}_1 | 0.391 | 0.024 | 0.208 | |
| P- \mathcal{H}_2 | 0.335 | 0.017 | 0.176 | |
| A- \mathcal{I}_1 | 0.924 | 0.618 | 0.771 | 0.65 |
| A- \mathcal{I}_2 | 0.864 | 0.658 | 0.761 | |
| A- \mathcal{E}_1 | 0.680 | 0.628 | 0.654 | |
| A- \mathcal{G}_1 | 0.742 | 0.689 | 0.715 | |
| A- \mathcal{H}_1 | 0.468 | 0.041 | 0.255 | |
| A- \mathcal{H}_2 | 0.453 | 0.034 | 0.244 | |
| A- \mathcal{UP} | 0.935 | 0.681 | 0.808 | |
| A- \mathcal{SL} | 0.836 | 0.538 | 0.687 | |
| A- \mathcal{CL} | 0.943 | 0.930 | 0.936 | |
| COB | 0.169 | 0.001 | 0.085 | |

Table 8.10: Efficiency of the various clustering algorithms used to generate the ontologies, with the efficiency computed with Equation 8.5.

Using Tables 8.7, 8.8, 8.9, and Equation 8.5, the efficiency of all the clustering algorithms were computed, and the results are displayed in Table 8.10. With an average efficiency of 0.65; agglomerative algorithms are, on average, twice as efficient as partitional ones. As one can see, the best performing algorithm is the agglomerative clustering with the *clink* criterion function. Its good results are explained by very good predication accuracy, along with the lowest clustering time. Remember that in OF's situation the similarity matrix S is pre-computed, which removes the $O(n^2)$ complexity step from agglomerative

filtering. To the opposite, COBWEB is the less efficient algorithm. Its low prediction accuracy and excessive clustering time on the MovieLens data set lead to an efficiency value close to 0.09.

8.3.4 Experiment VII - Behavior of multi-hierarchical ontologies

In this experiment, the objective is to test whether the multi-hierarchical ontology generated by Algorithm 9 does increase the recommendation accuracy or not. To test this aspect, the previous experiment with the Jester data set was reproduced, but using only Algorithm 9 to generate the ontology. Recall that Algorithm 9 generates a unique multi-hierarchical ontology from the users' preference profiles (see Section 6.4).

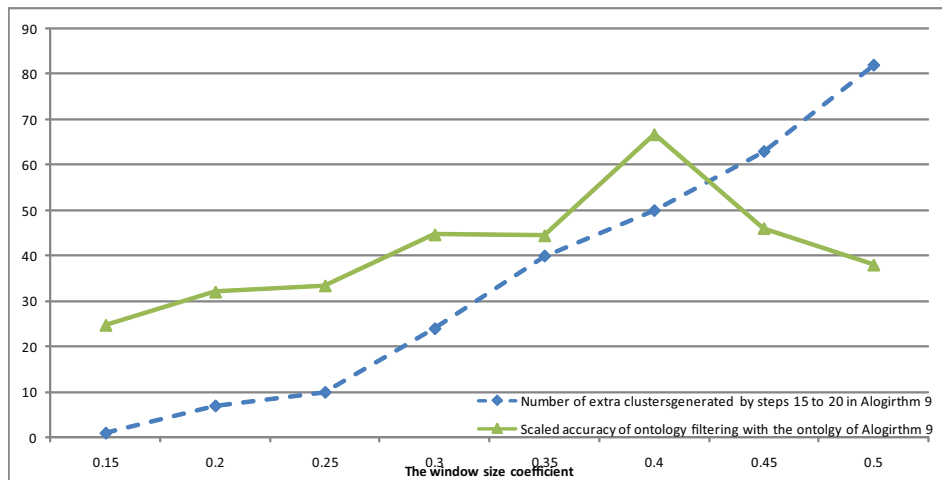


Figure 8.10: Number of extra clusters generated by steps 15 to 20 of Algorithm 9.

First, the experiment looked at the number of extra clusters generated by steps 15 to 20 of Algorithm 9. As expected, the dotted line in Figure 8.10 shows that by increasing the window size coefficient φ , many extra clusters are being created. Notice that following algorithm 9, one extra cluster implies that 2 clusters will have at least 2 parents. An interesting aspect to consider is whether keeping increasing the size of the window always improve the accuracy of the recommendation. The plain line in Figure 8.10 shows the average accuracy of the recommendation made by using Algorithm 9 for generating the ontology. The average accuracy was obtained by averaging the F1 values when respectively using 5, 10, 20, 40, 60, 80, and 100 leaf clusters in the ontology. Note also that the F1 metric values were scaled up to fit in the interval 0 to 90 by using the formula $y = 3000 * F1_{value} - 900$. The solid line clearly indicates that increasing the window size leads to better prediction accuracy. However, if the window becomes to big (i.e. $\varphi > 0.4$), then the structure becomes overloaded by inheritance edges, which significantly increases the search space and decreases the recommendation accuracy. Finally, notice that increasing the coefficient α will also increase the computational resources required to build the ontology. Thus, a tradeoff will need to be done between prediction accuracy and ontology quality.

The second part of the experiment looked at the overall recommendation accuracy of ontology filtering using the multi-hierarchical ontology compared to using a simple ontology generated from the agglomerative clustering. Figure 8.11 shows the prediction accuracy of the top-5 recommendation strategy obtained using the ontology filtering approach with different ontologies, and with different sizes of learning sets. The plain line models ontology filtering using Algorithm 9 to build the ontology, while the dotted line is OF that uses the classical agglomerative clustering with the *clink* criterion function for the ontology construction. These lines are respectively represented by *Algorithm_9_** and *A_clink_**, where * corresponds to the size of the learning set.

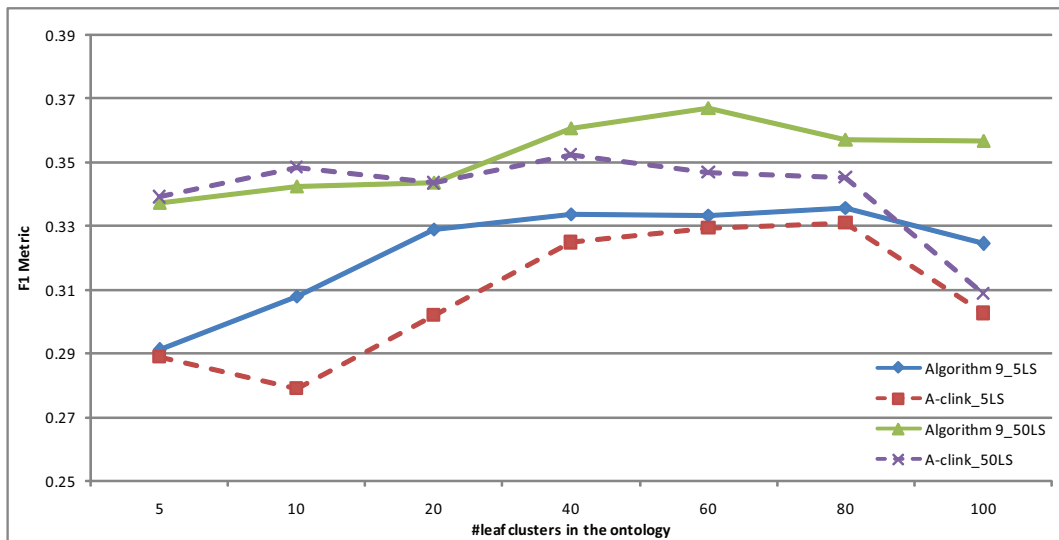


Figure 8.11: Accuracy of ontology filtering for the Jester data set when the ontology is generated from either the classical agglomerative clustering with *clink*, or with Algorithm 9 with φ set to 0.4.

These results tend to go in favor of OF's hypothesis, which states that a multi-hierarchical structure can further improve the recommendation accuracy compared to simple hierarchical one. As one can see, the ontology generated with the multiple inheritance algorithm reaches nearly always the best prediction accuracy, with the best improvements when the ontology contains at least 80 leaf clusters. When using traditional agglomerative clustering, the prediction accuracy fluctuates up and down depending on the number of leaf clusters, and significantly decreases when the ontology contains 100 leaves. This shows that if the granularity of the ontology is too small, then this leads to poor recommendation. This behavior is reduced with the multiple inheritance algorithm, which seems more robust to the number of leaf clusters. This can be explained by the fact that Algorithm 9 builds many extra inheritance edges, which means that the inference process has a higher probability of finding a shorter path between two given concepts. This shorter path will allow more score to be transferred between these concepts, which leads to a better inference process. Note that both approaches reaches their optimal values when the number of leaf clusters is between 40 to 60, which means that each cluster must contain between two to three instances.

This is consistent with the definition of a cluster that should contains a set of instances sharing identical features, rather that containing single instances

Even if these results show encouraging improvement when using multi-hierarchical ontologies, this does not compensate the explosion in complexity by considering sub-optimal clusters. Moreover, Algorithm 9 requires that the window size parameter φ is carefully selected. Thus, more work is required in order to better select these sub-optimal clusters to reduce the complexity. As a consequence, this dissertation leaves this as future work, and will carry on by using the ontologies generated using traditional clustering algorithm.

8.3.5 Experiment VIII - Ontology filtering vs collaborative filtering

Experiment V showed that ontology filtering was more robust to the similarity metric than the item-based collaborative filtering. Then, experiment VI showed that personalizing the ontology lead to better prediction accuracy compared to using the same ontology for users. Following these results, this last experiment focuses on whether the ontology filtering approach defined in details in Section 7.2 is in fact better or worse than classical item-based collaborative filtering.

For this experiment, ontology filtering is compared to the traditional item-based collaborative filtering using the adjusted cosine similarity for computing the similarity between pairs of items. Ontology filtering uses Algorithm 7 to generate all the ontologies, Algorithm 8 to select the best ontology to be used by each user, Algorithm 5 to build a complete user ontological profile, and then Algorithm 6 to recommend the top-N items. Note that this time, the hybrid score is computed using both the popularity and user's score (i.e: personalization coefficient $\rho = 0.5$).

The performance of CF greatly depends on the number of neighbors used to compute the prediction (Equation (2.25)). At the same time, the ontologies generated by Algorithm 7 will be different depending on the number of leaf clusters (i.e.: the threshold θ) that were specified. To include these aspects, the experiment was iterated using various values of leaf clusters and neighbors, where the number of leaf clusters is always set to the number of neighbors (i.e: $k = \theta$).

Figure 8.12 shows the accuracy of the OF and CF recommender systems on the Jester data set. The dashed lines represent collaborative filtering, while the plain lines model ontology filtering. As before, the notation $*-5LS$ and $*-50LS$ means that 5 and respectively 50 items were used to learn the model (i.e.: find the close neighbors for CF, and build the user's preference profile for OF). The x-axis shows the number of neighbors that were used for CF, which is also the same parameter used for the number of leaf clusters in Algorithm 1. The y-axis measures the accuracy of the recommendation using the F1 metric.

First, and most important, ontology filtering using the learnt ontologies performs much better than CF. The result is even more emphasized when very few ratings (only 5, *OF-5LS*) are used to learn the model. As a matter of fact, the improvement of OF is always significant (p-value $\ll 0.01$), whatever the number of leaf clusters used to build the ontology. Furthermore, OF with just 5 ratings to build the *UPP* performs better than CF with 10 times more data for its model construction. When 50 ratings are used for constructing the user's model, OF's accuracy remains better than CF's one, and with significant improve-

ment ($p\text{-value} \ll 0.02$), except when the number of leaf clusters is set to 20 ($p\text{-value} = 0.31$).

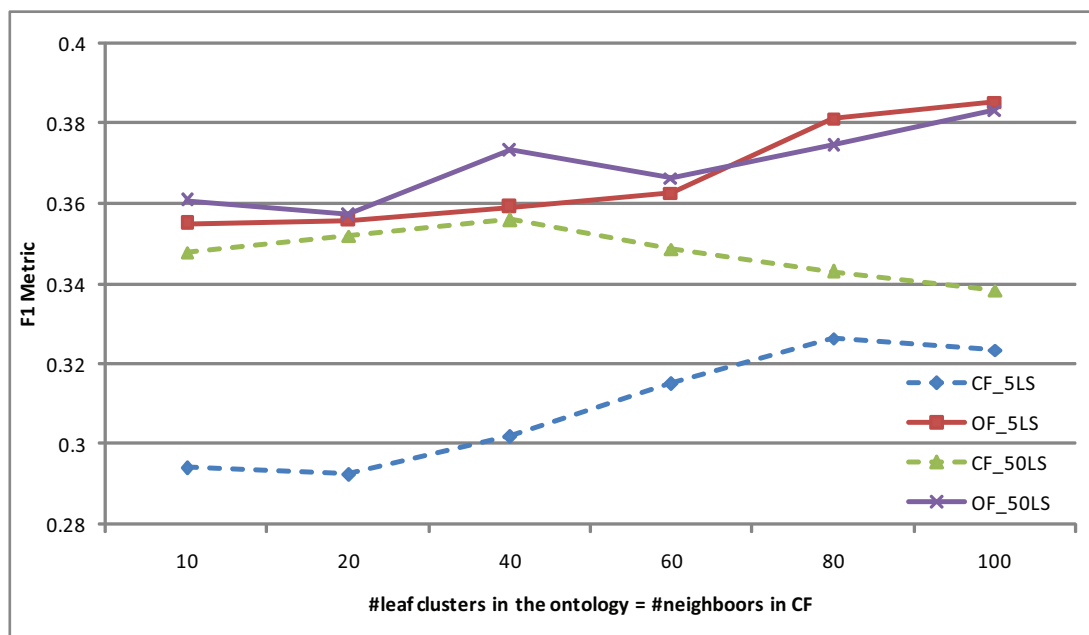


Figure 8.12: Accuracy of collaborative filtering and ontology filtering for the Jester data set. For each recommender system, the experiment was run once with only 5 ratings to learn the model (5LS), and then a second time with 50 rated items (50LS).

Notice that the accuracy of CF actually increases with the number of neighbors, and then decreases again. This is a well known result [Herlocker et al., 1999], and is due to two reasons. First, CF needs to have enough neighbors in order to correctly predict items. Second, if too many low correlated neighbors are included in the computation, then the accuracy decreases. This phenomenon seems amplified when 50 items are present in *LS*. OF is more robust to the number of clusters in the ontology than CF is to the number of neighbors. This is because OF inference is done using the closest element, while CF uses a neighborhood of items.

Figure 8.13 shows the accuracy of the same experiment, but performed on the MovieLens data set. Notice that for Jester, the maximum number of neighbors was set to 100 as Jester contained 100 items. However, MovieLens contains 1682 different movies, which can lead to a hierarchical tree with 1682 leaves. However, due to the sparsity of the user-item matrix, the similarities between some items could not be computed, which lead to only 1668 clusters. Again, OF performs much better than CF, and always shows significant improvement when 5 ratings are used to learn the model (with a $p\text{-value} < 0.01$). When 50 ratings are used, then the improvement remains statistically significant if the number of neighbors is less than 250 or bigger than 1000 (detailed $p\text{-value}$ can be found in Appendix E.3). Note that CF is again very dependant on the number of neighbors. If not enough good neighbors are selected, then the prediction accuracy remains poor. Inversely, if too many low correlated neighbors are considered, then the prediction accuracy decreases

[Herlocker et al., 1999]. Ontology filtering is more robust to the number of leaf clusters, which tends to show that the ontology and the inference mechanism are useful in the recommendation process.

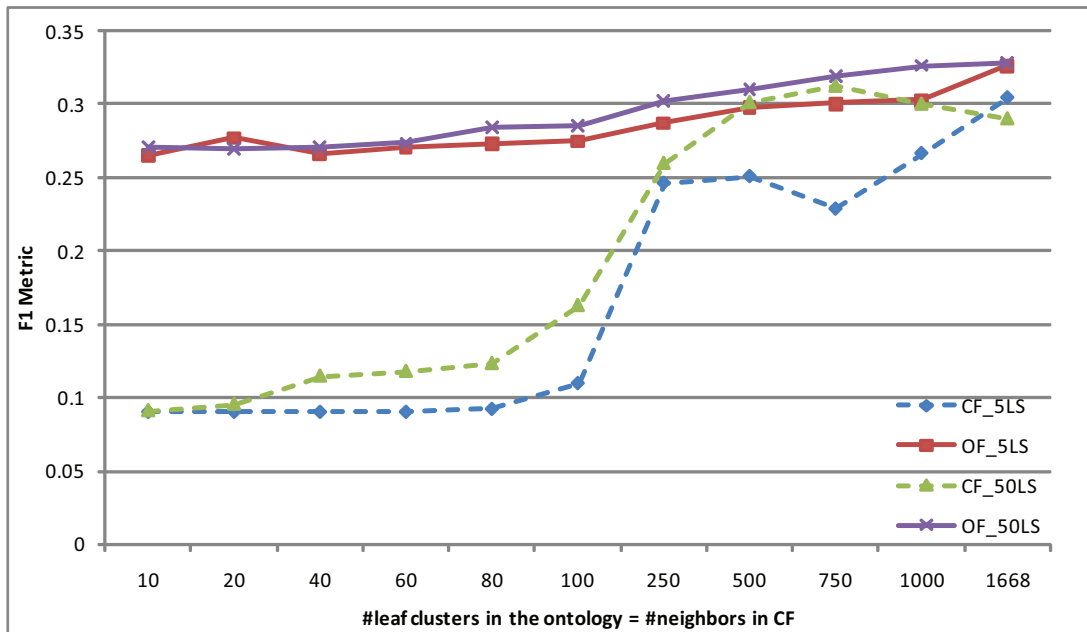


Figure 8.13: Accuracy of collaborative filtering and ontology filtering for the Movielens data set. For each recommender system, the experiment was run once with only 5 ratings to learn the model (5LS), and then a second time with 50 rated items (50LS).

When looking at the behavior of CF, the results are similar to the ones observed with Jester. However, the accuracy seems independent from the number of neighbors when it is less than 100. This can be explained by the fact that the MovieLens data set is much sparser than Jester. Thus, many more ratings are necessary to build a correct model of the users.

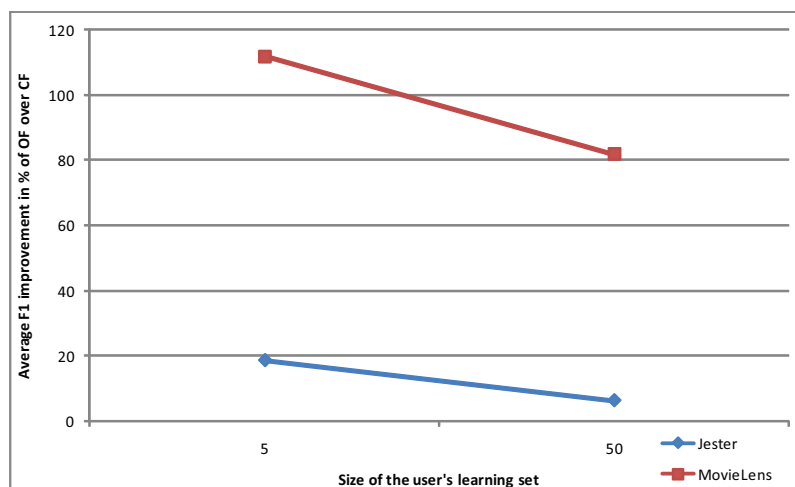


Figure 8.14: Average improvement in % of the F1 metric of OF over CF.

To summarize, Figure 8.14 contains the average improvement of the prediction accuracy of ontology filtering over collaborative filtering. These results are obtained by averaging the F1 value obtained by the different pairs of number of clusters and neighbors. When only 5 items are used to learn the user's preference model, ontology filtering increases the prediction accuracy by 111% for MovieLens and over 18% for Jester. When 50 items are inserted in the learning set, the improvement of ontology filtering decreases as collaborative filtering starts to have enough ratings to locate good correlated neighbors. Notice that, on average, the improvement on the MovieLens data set is more than six times bigger than the one obtained on Jester. This is explained by the fact that the MovieLens data set is much sparser than Jester, which implies that CF requires many ratings for finding the appropriate neighborhood. These results tend to suggest that ontology filtering is able to successfully restrict the search space.

Overall, it can be seen that OF is performing extremely well and always leads to significant improvement over CF when 5 ratings are used to build the preference model. Furthermore, OF with just 5 ratings in the learning set has a prediction accuracy that is nearly as good, sometime better, than the one of CF with ten times more training data. When 50 ratings are used to build the preference model, collaborative filtering performs better than with 5 ratings, but its accuracy remains worse than the one obtained by ontology filtering. This tends to suggest that the learnt ontologies are of good quality. One important fact that needs to be taken into account is that CF's accuracy tends to be proportional to the number of neighbors. When the size of the user's preference set increases, this leads to significant scalability problems. Ontology filtering is less critical to this problem as the inference is carried out from the closest concept, not on a neighborhood.

8.4 Discussion

These experiments bring more insight into the use of clustering algorithms to build ontologies, and the behavior of these ontologies in ontology filtering. From these, five main conclusions can be drawn.

1. The inference mechanism defined in Equation 5.8 can be derived to build a robust similarity function. Experimental results on WordNet and on the GeneOntology have shown that this function significantly outperforms state of the art similarity functions, and can better correlate with human judgments. The inference mechanism has also proven useful for estimating the utility values assigned to various themes. These experiments tend to suggest that the inference mechanism can correctly transfer preferences from one concept to another.
2. As expected, the adjusted cosine function is the function that produces the best item-to-item similarity. However, experiments have also shown that the difference with the classical Pearson or cosine functions is not statistically significant. Furthermore, ontology filtering seems much more robust to these three metrics than collaborative filtering.

3. When considering the prediction accuracy of partitional and agglomerative clusterings for building ontologies, there is no clear winner even though on average, partitional ones seem slightly better. However, when considering the efficiency of the algorithm, then the agglomerative clustering with *clink* criterion function significantly outperforms the others. Moreover, the accuracy will greatly be influenced by the number of leaf clusters set as parameter.
4. The intuition that using the same ontology for every user is sub-optimal seems correct, and explains why Algorithm 8 that personalized the ontology based on the user's preferences leads to significant increase in prediction accuracy.
5. The second intuition of considering sub-optimal clusters in order to generate multi-hierarchical ontologies seems also verified. However, even if experimental results have shown some improvement in the recommendation accuracy, this is at a greater cost in computational complexity which explodes with the window size φ .

Beside significantly improving the prediction accuracy over traditional item-based collaborative filtering, ontology filtering has other advantages. Following the previous analysis in Section 2.7, the advantages and drawbacks of ontology filtering are summarized in Table 8.11.

| Approach | Input Data | Advantages | Disadvantages |
|----------|-----------------------|--|---|
| OF | Ratings on some Items | No domain knowledge needed Low cognitive requirement Good prediction accuracy Implicit feedbacks sufficient Good domain discovery Preserve user's privacy Partially solve cold-start More scalable than CF ³ | Training data Shilling problem ¹ Latency problem ¹ Ontology construction Scalability ² |

Table 8.11: Brief tradeoffs analysis of ontology filtering; ¹ when learning the ontologies with Algorithm 7, ² when using multi-hierarchical ontologies, and ³ when using a small set of hierarchical ontologies.

As with collaborative filtering, ontology filtering requires little domain knowledge and low cognitive resources from the user, as her preferences are expressed as rated items. However, ontology filtering has been found to be more accurate on both Jester and MovieLens, while keeping a high degree of novelty. This goes in line with [Ziegler et al., 2005] who suggested to use an ontology in order to increase diversity in the recommendation. Furthermore, experimental results show that ontology filtering performs better than collaborative filtering that uses 10 times more data to learn the user's preference model. Ontology filtering turns out to be also more scalable than collaborative filtering, as the inference is done from the closest concept rather than from a neighborhood. The last experiment revealed that ontology filtering took on average less 15 ms to come up with the top-5 recommendations, which is over 10 times less than collaborative filtering. However, it is fair to point

out that this fact does not hold in two situations. First, if the set of available eCatalog ontological models is too big, then the search of the optimal one for a given user will greatly increase the computational time. Second, the search of the lowest common ancestor is linear in complexity only if hierarchical ontologies are considered, not with DAG structured ones. Another advantage of ontology filtering is to significantly reduce the cold start problem. This is due to the fact that an ontology adds a structure on top of the eCatalogs, which limits the space of possible items to search for. Notice that the recommendation does need to be done on the server, but can actually be done by the clients if it has the ontologies. This potentially solves the privacy problem as the system will not know the user's preferences, and increases further the scalability of the system as the computation is distributed. One could argue that the privacy problem could also be solved for item-based collaborative filtering if the item-matrix S is sent to the user. This would indeed solve the privacy of the user, but could violate the privacy of others, as the matrix R could be reconstructed from S , if sufficient data is known. Remember that ontologies used in ontology filtering do not contain pairwise similarities but just a-priori scores. Thus, it makes it much harder to reconstruct the matrix R , as a malicious user would just have partial ordering of the items.

Unfortunately, as the ontologies are learnt using previous users' ratings, ontology filtering can also suffer from shilling attacks and sparsity issues. Similarly, the latency issue remains a problem with learnt ontologies as no ratings are available. When items can be described by attributes (i.e.: explicit features), then unrated items can be predicted by associating them to items with a score that have similar attributes. These three problems remain open and are left as future work. Unfortunately, the ontology construction will remain a handicap for ontology filtering as this structure takes time to build. Nevertheless, this construction time can be reduced by using Algorithm 7, but it will never be insignificant. Finally, note that ontology filtering can also suffer from scalability issues when using ontologies with DAG structure, as finding the lowest common ancestor in a DAG is $\mathcal{O}(n^{2.688})$.

Chapter 9

Conclusions

*All goods things come to an end, eventually.
A common say.*

9.1 Scope

With the rise of the internet, people are becoming overwhelmed by information. To help the user in this process, researchers have come up with recommender systems. A recommender system is a computer program that helps people find relevant items based on the person's preferences and others. Nowadays, two kinds of recommender techniques are becoming increasingly popular in eCommerce Sites: collaborative filtering and the preference based approach.

Collaborative filtering (CF) recommends items to the user based on the experiences of others. Thanks to Amazon.com, this technique is now the most popular approach as it has shown good recommendation accuracy in practice. The fundamental assumption behind CF is that similar users like similar items. Thus, the ability of CF to recommend items depends on the capability of identifying a set of similar users. Furthermore, it does not build an explicit model of the user's preferences. Instead, preferences remain implicit in the ratings that the user gives to some subsets of products. Despite being popular, CF has two well known problems: the cold-start problem and the scalability. The former problem occurs when a new user enters the system and is asked to rate many items, which is usually more than what a user can or is willing to rate. The latter problem arises because the computation of the neighborhood of similar items grows with the number of items and users.

The other technique is the preference-based approach, (PBA). Here, a user is asked to express explicit preferences for certain *attributes* of the product. If preferences are accurately stated, then multi-attribute utility theory provides methods for finding the most preferred product even when the set of alternatives is extremely large and/or volatile. Thus, PBA does not suffer from problems encountered by collaborative filtering. However, the big drawback of preference-based methods is that the user needs to express a potentially quite complex preference model. This may require a large number of interactions, and places a higher cognitive load on the user since she has to reason about the attributes modeling the products. To be able to reason over the items, MAUT also requires that products

are modeled by a set of well defined attributes. Depending on the domain of application, attributes are very hard to express and frequently change. This partially explains why MAUT is only used in simple domain such as notebooks.

This dissertation believes that the problems faced by the above approaches are due to the lack of appropriate model for representing the items, and the preference elicitation overload. Thus, a new technique called ontology filtering is proposed that can overcome most of the problems faced by previous approaches, while achieving a better prediction accuracy than the item-based collaborative filtering. The main novelties of this technique are to model the items by ontology, and to use this ontology to infer missing preference values in order to build a complete user preference model.

To illustrate ontology filtering, a prototype has been implemented that can recommend jokes to the user. Appendix C contains a full scenario, along with some illustrations of this prototype.

9.2 Contributions

This section outlines the main results and contributions of this thesis.

Modeling eCatalogs with an Ontology - The idea of using a taxonomy for modeling the items of an eCatalog is not new and was introduced by [Middleton et al., 2004]. However, four aspects make the ontology used by this dissertation very different from existing work.

1. In previous work, authors use simple taxonomies that are assumed to be freely available to the system. This work makes no such assumption, and proposes an algorithm that can learn these ontologies automatically, and without the user's intervention. Moreover, more complex structures such as directed acyclic graph are also considered.
2. The ontology does not need to have the features explicitly modeled by the edges, but instead can remain implicit. This allows to model complex domains such as joke or wines.
3. Ontology filtering is capable of extracting the information contained in the topology of the ontology. This information is then used to compute the a-priori score of each concept, and infer missing information through an inference mechanism.
4. The ontology is used directly for making the prediction, while other researchers combine it with collaborative filtering to fill in missing ratings and reduce the sparsity of the user-item matrix R .

Extracting the information of an ontology : This dissertation shows that, under three fundamental assumptions, the topology of the ontology contains some information. This information can be extracted and used to compute the a-priori score of a concept c , $APS(c)$, which happens to be equal to one over the number of descendants of c plus two.

Using the APS and the ontology to infer missing preferences : Users express their preferences by rating a set of items. In the ontology model, these items are instanced of some concepts. Thus, these preferences can be used to compute the score of a concept. Given the a-priori scores and a score, this dissertation defined an inference mechanism that can infer the score of any concept in the ontology given one unique score.

A new similarity function is derived from the inference mechanism, which allows to compute the pairwise similarity of any pair of concepts in the ontology. Moreover, experimental results have shown that this new similarity function outperforms state of the art metrics, and better correlates with human judgements on the WordNet ontology.

Learning the ontologies automatically : This dissertation acknowledges the fact that assuming the existence of an ontology is unrealistic. As a consequence, two algorithms for learning these ontologies automatically are introduced. The first one allows ontology filtering to build a set of 15 distinct hierarchical ontologies, while the second extends classical agglomerative clustering and builds a multi-hierarchical ontology.

Personalizing the ontology : Given a user u , it is commonly agreed in collaborate filtering that some neighbors correlate more than others with u . Similarly, this dissertation also believes that some ontologies are better for representing the user's preferences than others. That is why an algorithm is introduced that can select which ontology to use, given the user's preferences.

An ontology filtering architecture has been presented that combines all of the previously stated algorithms in order to generate recommendations to the user based on her preferences. Moreover, the resulting system allows to reach a much higher recommendation accuracy than collaborative filtering, even when 10 times less data about the user is known.

Experimental results are provided that give more insight into ontology filtering and clustering algorithms. Moreover, these experiments allow to validate the ontology filtering on data sets containing real user judgements.

In short, ontology filtering allows to build better recommender systems that can recommend items to the user with high accuracy, while reducing the number of questions asked to the user. Ontology filtering has another three significative advantages over collaborative filtering:

1. Scalable - ontology filtering scales better to big problems as the inference mechanism is done from the closest concept with some preference, rather from a neighborhood. Note also that clustering algorithms such a K-means partitional clustering can cluster thousands of items in minutes.
2. No parameter training - ontology filtering does not have any parameter that requires training, and can be used directly on any data set that contains rated items. Note that the number of leaf concepts can be set to the maximum number of items.

3. Explanation - ontology filtering can easily explain how recommendations are computed as it uses the closest concept to the user's preferences. This allows the system to build trust with its users, and increases the user's satisfaction and loyalty.

9.3 Limitations

Obviously, this dissertation does not claim that ontology filtering can solve *all* the recommendation problems. Limitations of the presented approach can be summarized in the following way:

Fundamental Assumptions : Ontology filtering makes three fundamental assumptions about the score of a concept. First, the score depends only on the features making an item. Second, each feature contributes independently to the score. Third, features that are unknown make no contribution to the score. The first assumption is not very restrictive as the features can be captured implicitly in the edges. However, the other two assumptions induce the following limitations:

1. *feature independence* eliminates the inter-dependance between features, and allows the score to be modeled as the sum of the scores assigned to each feature. Thus, modeling features such as the price of an item could be problematic as the price usually depends on all the other features. Note however that the multi-attribute utility theory makes an even stronger assumption to build the additive value function [Keeney and Raiffa, 1993] [Payne et al., 1988].
2. *low risk domain* is where ontology filtering can be used due to the third assumption. In other domain such as gambling, the third assumption will not hold as people are risk seeking rather than risk-averse. Financial domain is another example where ontology filtering will probably not be very useful as the risk of failure is very high and people are ready to spent a significative amount of time expressing their preferences.

The Ontology construction can take a significant amount of time, especially if hand-crafted. To reduce this problem, ontology filtering allows ontologies to be learnt automatically in order to build the eCatalog ontological model. However, even if these ontologies can be learnt off-line, the clustering algorithms can take significant amount of time, if the system contains millions of items and users. Fortunately, this time can be greatly reduced by parallelizing the computation of the ontologies.

Multiple-Hierarchy ontology can increase the recommendation accuracy but at a great cost in computational time. When considering directed acyclic graph structure, two problems arise:

1. *ontology construction time* is significantly longer as more than one cluster can be merged (or split) at each iteration.

2. *finding the lowest common ancestor* is not trivial in a DAG as there can be more than one path connecting two concepts. Moreover, the best algorithm for finding a LCA in a DAG has a complexity of $\mathcal{O}(n^{2.688})$.

The a-priori score is computed based on the idea that users are risk averse, and that the score of the leaves are uniformly distributed in the interval $[0, 1]$. In practice however, not all the scores of the leaves will be uniformly distributed, and some will be higher than others. This uneven distribution also implies that the ontology should be balanced. Obviously, balanced ontologies rarely exist, but clustering algorithms minimize this problem by trying to have equal number of instances in each leaf cluster.

9.4 Future research

The above limitations indicate that more research can be undertaken to improve ontology filtering. Given these limitations, this section highlights the possible new research directions.

Relaxing the assumptions would cover wider domains of application and reduce the limitations of the three fundamental assumptions. For example, it would be very interesting to consider the situation where feature dependance is taken into account.

Improve the ontology construction by considering more clustering algorithms. This dissertation has used basic partitional and agglomerative clustering algorithms, but more complex algorithms such as fuzzy k-means or Possibilistic K-means would allow to consider non-linear properties and softer class membership. Incremental clustering algorithms would also be useful to reduce the computational time by reusing existing ontologies.

Adding more semantic relations : Currently, only inheritance edges are being considered. However, as shown by WordNet, other relations such as meronym and holonym have proved useful. Note that adding extra relations will also influence the inference mechanism.

Improving the a-priori score of concepts would also increase the recommendation accuracy as the inference mechanism uses them to estimate propagation coefficients. For example, two assumptions are used to compute these a-priori scores. First, the score of the leaves concepts are all uniformly distributed. Second, the user is risk averse, which implies that for a user to like a concept c , she must like all the instances of the descendants of c and of c itself. Thus, the APSs could be improve by relaxing these assumptions.

9.5 Conclusion

Currently, there are two widely used techniques to solve the recommendation problem. The first approach is collaborative filtering that recommends items based on the experience of

similar users. The second one is the preference-based approach that models items by its features, and uses only the user's preferences to recommend her items.

Despite being very popular, these recommender systems fail to achieve high recommendation accuracy in eCommerce environments; especially when users' preferences are rare. Our analysis shows that it is due to two fundamental problems. First, current recommender systems use inappropriate models of the items, which leads to unconstrained search space. Second, recommender systems must elicit too many preferences from the user in order to build the user's preference profile.

This dissertation proposes the ontology filtering approach that can overcome most of the problems faced by previous approaches, while achieving better prediction accuracy than item-based collaborative filtering on Jester and MovieLens. The intuition behind ontology filtering is that the information captured by the topology of the ontology can be used to estimate missing preferences. The main novelties of this technique is to model the content of the eCatalog and infer missing preferences using the ontology, and use the inferred information in order to directly recommend items to the user.

Appendix A

Classification of recommender techniques

To ease the reading of this appendix, Table A.1 gives the section number where the approach in table A.2 has been discussed.

| Recommendation Technique | Section where this technique is introduced |
|--------------------------------------|--|
| Non-Personalized Recommender System | 2.2.1 |
| Content-Based Recommender System | 2.3.1 and 2.3.2 |
| Preference-Based Recommender System | 2.3.3 |
| Memory-Based Collaborative Filtering | 2.4.1 |
| Model-Based Collaborative Filtering | 2.4.2 |
| Knowledge-Based Recommender System | 2.5 |

Table A.1: Sections where the various approaches in Table A.2 have been introduced.

Table A.2 shows the correspondences in terminology between the authors that have been studied in Section 2.2.

| Dissertation | Memory-Based | Model-Based | Knowledge-Based | Content Based | Preference Based | Non Personalized |
|----------------------------------|--------------------------|------------------------------|-----------------|--------------------------|------------------|-------------------|
| [Schaffer et al., 1999] | Item to item Correlation | People to People Correlation | | Item to item Correlation | Attribute Based | Aggregated Rating |
| [Terveen and Hill, 2001] | Collaborative Filtering | Collaborative Filtering | | Content Based | Content Based | |
| [Burke, 2002] | Collaborative Filtering | Collaborative Filtering | Knowledge-Based | Content Based | Utility-based | |
| [Adomavicius and Tuzhilin, 2005] | Collaborative Filtering | Collaborative Filtering | Knowledge-Based | Content Based | Content Based | |

Table A.2: Overview of the techniques used in Recommender Systems and their name given by various authors.

Appendix B

Transitivity of OSS

To verify that the distance measure satisfies the triangle inequality, consider the transfer from concept x to z (Figure B.1 exported from Section 3.3.2), and the additional concepts s and t .

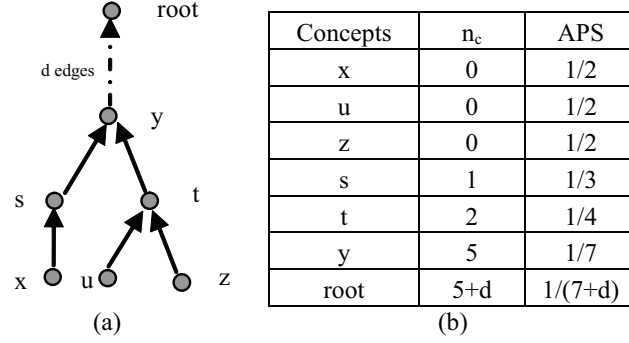


Figure B.1: (a) a simple ontology λ and its APSs (b).

This dissertation believes that the primary objective of a similarity measure is to simulate a user's behavior and closely correlate with it. In particular, a quantitative measure of similarity should express the ratio of numerical scores that may be assigned to each concept. The score could reflect how much an item is preferred, or how friendly it is to the environment.

Assume first that s is a node on the path from x to y . Then if s is part of the upward path from x to y , equation (5.2) implies that $[T(x, s)] = APS(s)/APS(x)$ and $[T(s, y)] = APS(y)/APS(s)$. Furthermore, and because $[T(x, y)] = APS(y)/APS(x)$, it is easy to see that:

$$\begin{aligned} -\log([T(x, y)]) &= -\log([T(x, s)] \times [T(s, y)]) \\ &= -\log([T(x, s)]) - \log([T(s, y)]) \end{aligned} \quad (\text{B.1})$$

As a consequence, $D(x, y) = D(x, s) + D(s, y)$ and thus the triangle inequality holds as $D(x, z) = D(x, s) + D(s, y) + D(y, z)$. If t is part of the downward path from y to z , then we get that $[T(y, t)] = 1/(1 + 2\beta(t, y))$ and $[T(t, z)] = 1/(1 + 2\beta(z, t))$. By definition, $[T(y, z)]$ is equal to $1/(1 + 2\beta(z, y))$, and Equation 5.5 implies that $\beta(z, y) = \beta(z, t) + \beta(t, y)$. Thus, we get the following equation:

$$\begin{aligned}
-\log(\lfloor T(y, z) \rfloor) &= -\log\left(\frac{1}{1 + 2\beta(z, y)}\right) \\
&\leq -\log\left(\frac{1}{1 + 2\beta(z, y) + 4\beta(z, t)\beta(t, y)}\right) \\
&= -\log\left(\frac{1}{1 + 2\beta(z, t)} \times \frac{1}{1 + 2\beta(t, y)}\right) \\
&= -\log(\lfloor T(y, t) \rfloor) - \log(\lfloor T(t, z) \rfloor)
\end{aligned} \tag{B.2}$$

which shows that the triangle inequality also holds when t is on the downwards path connecting y and z .

Now consider the case where s is not on the path from x to z . Due to the graph structure, the paths from x to s and from s to z will both contain a common part between a node s and a node d that is on the path from x to z . Since all transfers are ≤ 1 , this part can only increase the combined distance, so the triangle inequality holds.

Appendix C

Ontology filtering: a scenario

To illustrate the ontology filtering approach, a prototype has been constructed that closely follows the architecture defined in chapter 7. Using the Jester dataset as eCatalog, the prototype recommends 5 jokes to the user based on her preferences that were either directly elicited or obtained as feedback. Formally, the recommender problem for this scenario is as follows.

Given a collection of 100 jokes, recommend 5 items to the user Vincent based on his preferences.

C.1 The scenario

Vincent¹ is a computer scientist with a big sense of humor, and loves starting a day of work with a good joke. The problem that Vincent faces is the frustration against classical humor web sites that do not take his (engineer) sense of humor into consideration. Fortunately, Vincent heard about this new joke recommender system while at a conference in Hyderabad, India. After talking to some researchers there, Vincent was promised an access to the web site as soon as it would be in a stable version. Couple of weeks later, Vincent receives an email with the address of the recommender system and the password. Let's imagine that Vincent tries to use the web site to get some recommendations.

Vincent starts by opening the web browser of his computer, and types in the URL of the ontology filtering recommender system. For ontology filtering to retrieve the right preference profile, Vincent must identify himself with the password he received in the email (Figure C.1). Once Vincent is identified, the recommender systems notices that he has no preference profile. Thus, OF starts the elicitation process by asking Vincent to either rate joke 17, or to enter a known joke (Figure C.2). As Vincent loves joke 17, he gives it 5 stars.

After answering 5 elicitation questions, ontology filtering shows the first recommendation that it has computed based on the 5 elicited preferences. Figure C.3 shows that OF recommends joke 89 in less than 5 milliseconds. Under the joke, Vincent can find a brief textual explanation of why this joke has been recommended to him. From this, he learns that joke 89 is instanced of concept 1000021, and that the hybrid score of this concept is 0.7019 (i.e.: the average of Vincent's score and the popularity score). However, what is

¹Any resemblance with the author of this dissertation will be total coincidence

more useful for Vincent is to know that this joke was recommended to him because he rated joke 11 with 5 stars. This means that ontology filtering started the inference from the concept 100010 representing joke 11.

As *Vincent* is a nosy engineer, and to be sure that it is really joke 11 which is the reason of this recommendation, *Vincent* decides to modify his preferences by clicking on the *Modify my preferences* link located to the left of the recommended joke. As a result, ontology filtering displays all of Vincent's preferences, with the option to either delete them or modify them (Figure C.4). Vincent selects joke 11 and updates the rating from 5 to 3 stars (Figure C.5). Figure C.6 shows the new recommendation that is generated from the updated preferences. As expected, the recommendation changes, and the new joke gets recommended because of the preference *Vincent* had on joke number 10.

Vincent remembers that one of the researchers at the conference told him that ontology filtering is capable of personalizing the ontology based on the user's preferences. He also remembers that the trick to identify the ontology is to look at the first digit of the concept. For example, the recommendation in Figure C.6 is instanced of concept 100065, which means that it was derived using the first ontology. As computer scientists cannot stop messing up things that work (notice how the joke that was recommend in C.6 matches an engineer's sense of humor), *Vincent* changes all of his preferences that had 5 stars rating to 4 stars. Figure C.7 confirms the updates, and also displays the updated preferences.

To get some more recommendations, *Vincent* clicks on the *Find recommendations* links located to the left Figure C.7. Less than 5 milliseconds later, ontology filtering produces a new recommendation (Figure C.8). Again, *Vincent* is impressed that the system is capable of predicting him jokes with engineers in it. But *Vincent* also notices that this time the recommendation was made using the third ontology, and that the score was inferred from concept 300082.

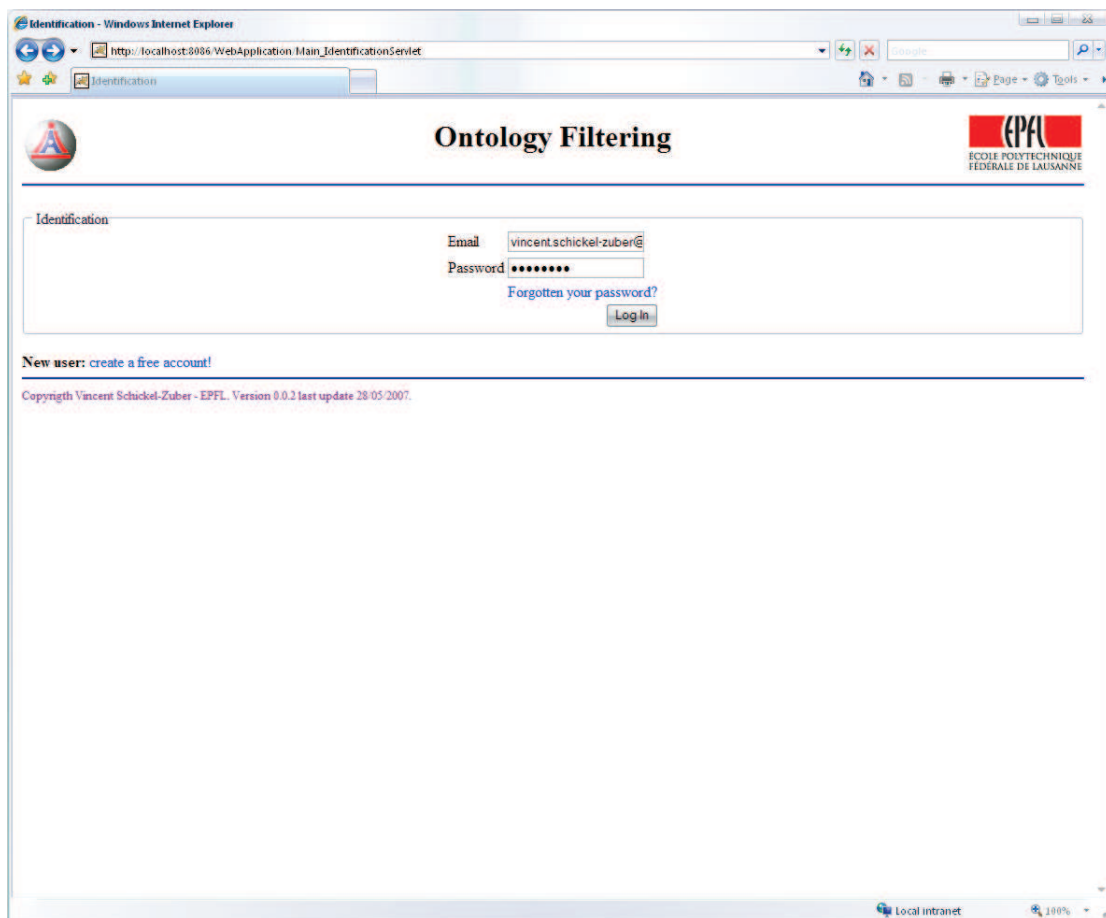


Figure C.1: The identification page where user *Vincent* identifies himself.

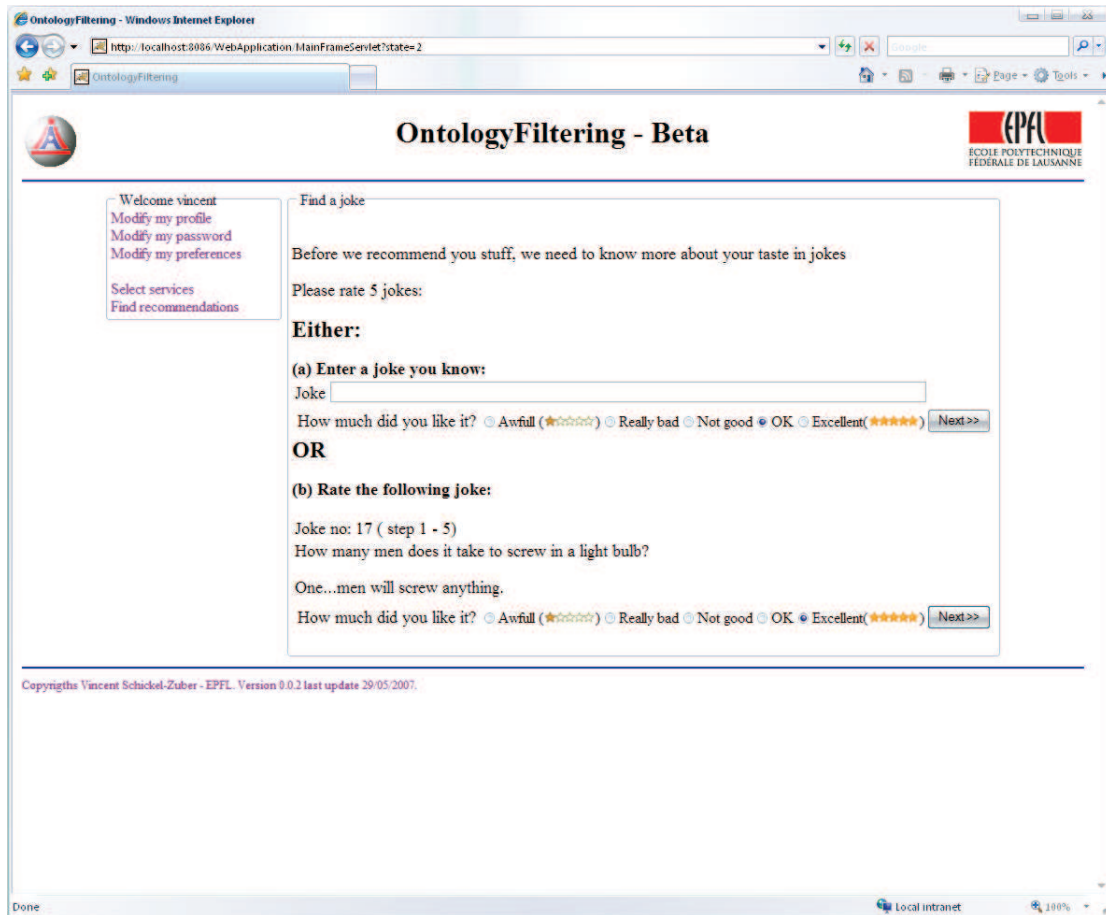


Figure C.2: Ontology Filtering asks *Vincent* to either rate joke 17 or enter a joke of his choice. *Vincent* loves joke 17 and gives it 5 stars.

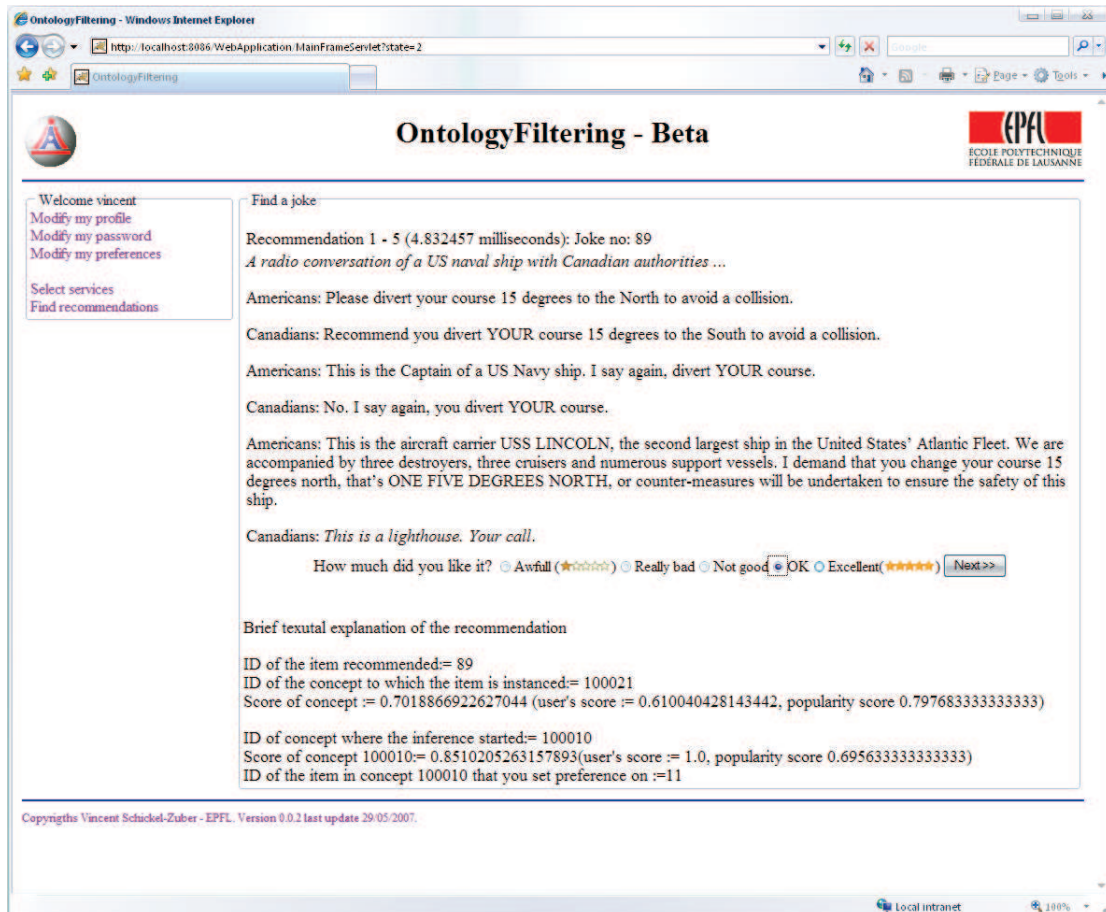


Figure C.3: The joke that is recommended to *Vincent* once the elicitation process is over. Joke 89 is recommended to *Vincent* as it is very close to joke 11 that he previously rated with 5 stars.

OntologyFiltering - Beta

Welcome vincent
[Modify my profile](#)
[Modify my password](#)
[Modify my preferences](#)
[Select services](#)
[Find recommendations](#)

Your preferences
Please select which preference you want to modify:

| DELETE | Items ID | Name | Description | Ratings |
|--------------------------|----------|-------------|--|--------------------------|
| <input type="checkbox"/> | 17 | Joke no: 17 | How many men does it take to screw in a light bulb? One...men will screw anything. | 5 Update |
| <input type="checkbox"/> | 92 | Joke no: 92 | Early one morning a mother went to her sleeping son and woke him up. "Wake up, son. It's time to go to school." "But why, Mama? I don't want to go to school." "Give me two reasons why you don't want to go to school." "One, all the children hate me. Two, all the teachers hate me," "Oh! that's no reason. Come on, you have to go to school," "Give me two good reasons WHY I should go to school!" "One, you are <i>fifty-two</i> years old. Two, you are the <i>principal</i> of the school." | 5 Update |
| <input type="checkbox"/> | 11 | Joke no: 11 | Q. What do a hurricane, a tornado, and a redneck divorce all have in common? A. Someone's going to lose their trailer... | 5 Update |
| <input type="checkbox"/> | 10 | Joke no: 10 | Two cannibals are eating a clown, one turns to other and says: "Does this taste funny to you?" | 5 Update |
| <input type="checkbox"/> | 34 | Joke no: 34 | Out in the backwoods of some midwestern state, little Johnny arrives at school an hour late. Teacher: "Why are you so late, John?" Johny: "My big brother got shot in the ass." (the teacher corrects his speech) Teacher: "Rectum." Johnny: "Wrecked him!? Hell, It damn near killed him!" | 1 Update |

Copyrights Vincent Schuckel-Zuber - EPFL. Version 0.0.2 last update 29/05/2007.

Figure C.4: Ontology filtering shows *Vincent's* preferences.

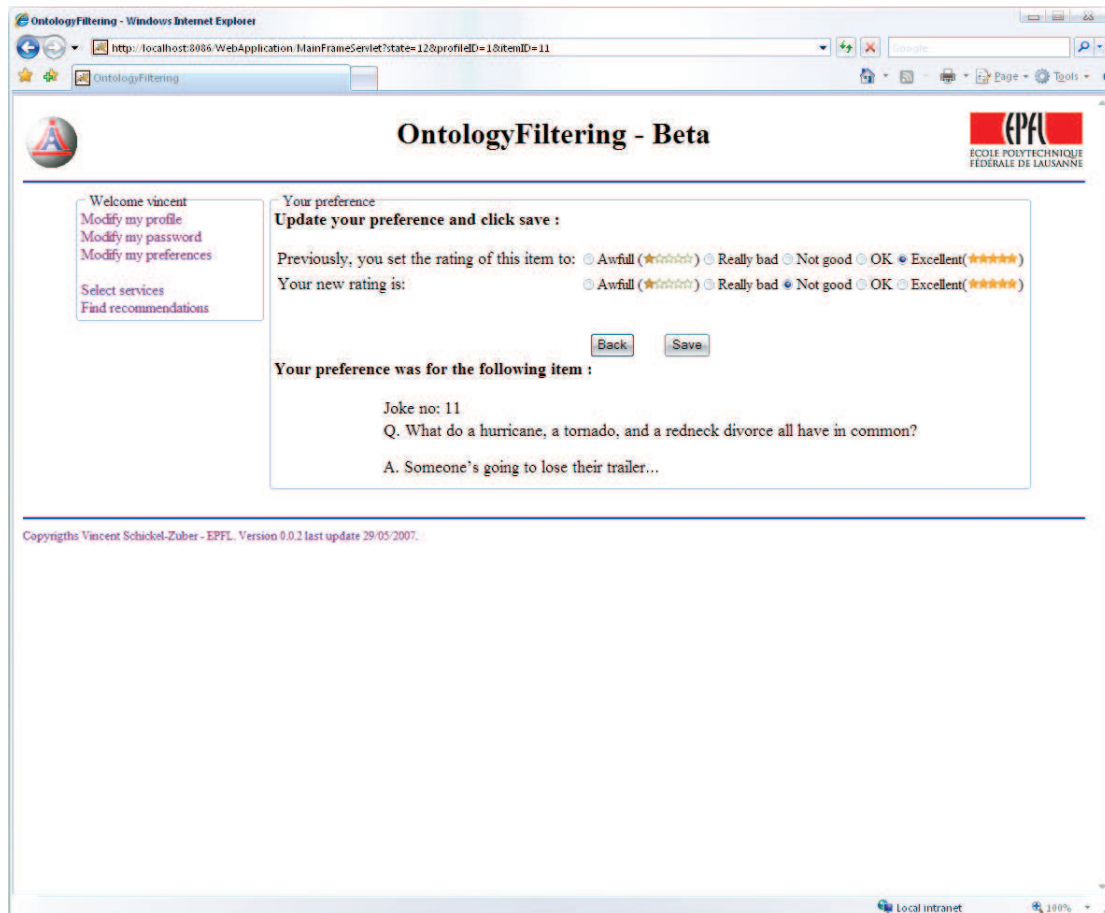


Figure C.5: *Vincent* updates his preferences on joke 11 by changing the rating from 5 to 3 stars.

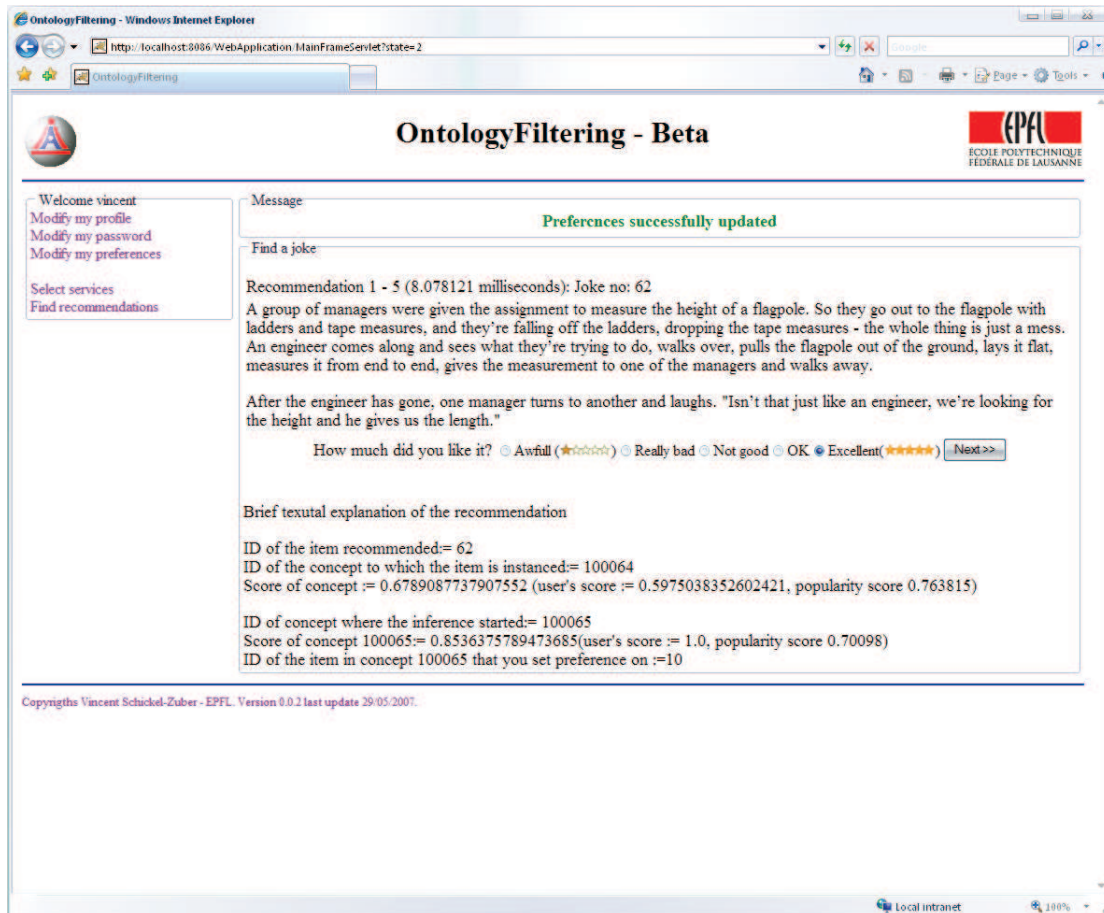


Figure C.6: The new recommendation that is made once *Vincent* has updated the rating of joke 11.

OntologyFiltering - Beta

Welcome vincent
[Modify my profile](#)
[Modify my password](#)
[Modify my preferences](#)
[Select services](#)
[Find recommendations](#)

Message
Preferences successfully updated

Your preferences
Please select which preference you want to modify:

| DELETE | Items ID | Name | Description | Ratings |
|--------------------------|----------|-------------|---|---|
| <input type="checkbox"/> | 17 | Joke no: 17 | How many men does it take to screw in a light bulb? One...men will screw anything. | 4 <input type="button" value="Update"/> |
| <input type="checkbox"/> | 10 | Joke no: 10 | Two cannibals are eating a clown, one turns to other and says: 'Does this taste funny to you?' | 4 <input type="button" value="Update"/> |
| <input type="checkbox"/> | 92 | Joke no: 92 | Early one morning a mother went to her sleeping son and woke him up. "Wake up, son. It's time to go to school." "But why, Mama? I don't want to go to school." "Give me two reasons why you don't want to go to school." "One, all the children hate me. Two, all the teachers hate me," "Oh! that's no reason. Come on, you have to go to school," "Give me two good reasons WHY I should go to school?" "One, you are <i>fifty-two</i> years old. Two, you are the <i>principal</i> of the school." | 4 <input type="button" value="Update"/> |
| <input type="checkbox"/> | 62 | Joke no: 62 | A group of managers were given the assignment to measure the height of a flagpole. So they go out to the flagpole with ladders and tape measures, and they're falling off the ladders, dropping the tape measures - the whole thing is just a mess. An engineer comes along and sees what they're trying to do, walks over, pulls the flagpole out of the ground, lays it flat, measures it from end to end, gives the measurement to one of the managers and walks away. After the engineer has gone, one manager turns to another and laughs. "Isn't that just like an engineer, we're looking for the height and he gives us the length." | 4 <input type="button" value="Update"/> |
| <input type="checkbox"/> | 11 | Joke no: 11 | Q. What do a hurricane, a tornado, and a redneck divorce all have in common? A. Someone's going to lose their trailer... | 3 <input type="button" value="Update"/> |

Done Local intranet 100%

Figure C.7: Vincent updates all of his preferences with 5 stars to 4 stars.

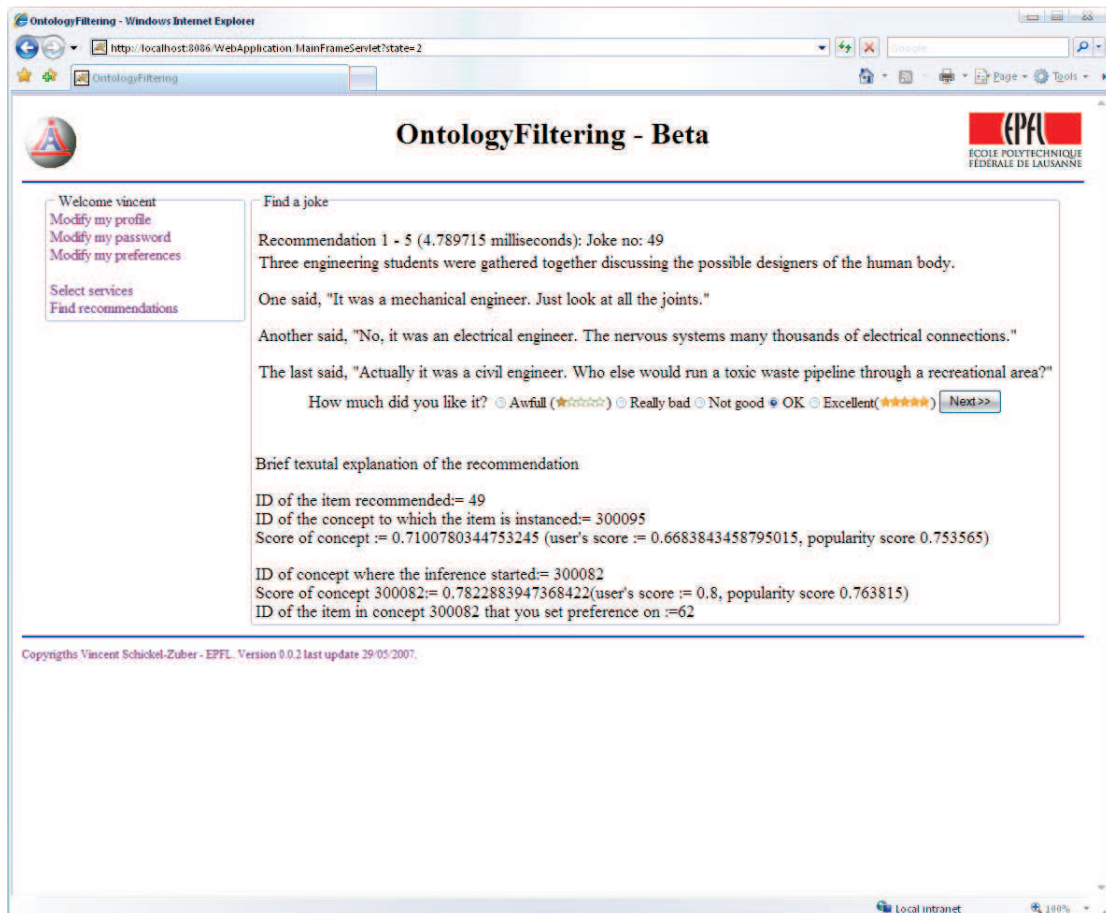


Figure C.8: Final recommendation made when no more preferences have a 5 star ratings. Note that the recommendation is now made using the third ontology, where the ontology is identified by the first digit of the identifier given to a concept.

Appendix D

Existing ontologies

D.1 WordNet

WordNet¹ is a large semantic lexicon of the English language, developed at the Cognitive Science Laboratory of Princeton University. It groups English words into sets of synonyms called *synsets*, which are linked together with various semantic relations. These relations can be of four different types:

- *Hypernym* - The generic term used to designate a whole call of specific instances. Y is a hypernym of X if X is a kind of Y.
- *Hyponym* - The specific term used to designate a member of a class. The hyponym relations is the inverse of the hypernym. X is a hyponym of Y if X is a kind of Y.
- *Meronym* - The name of a constituent part of, the substance of, or a member of something. X is a meronym of Y if X is part of Y
- *Holonym* - The name of the whole of which the meronym names a part. Y is a holonym of X if X is a part of Y.

Figure D.1 contains an extract of WordNet for the synset red-wine. These relations can be summarized as follows. Note that the term *IS-A* is commonly used to denote *kind-of*.

| Relation | Meaning | Example | Inverse relation |
|----------|---------|----------------------------------|------------------|
| Hyponym | Kind-of | red-wine is a kind-of motor wine | Hypernym |
| Meronym | Part-of | red-wine is part-of sangria | Holonym |

Table D.1: Summary of the WordNet relations.

As of 2007², WordNet contains over 155'000 words organized in over 117'000 concepts for a total of 207'000 word-sense pairs. The words are separated into nouns, verbs, adjectives, and adverbs because they follow different grammatical rules.

¹<http://wordnet.princeton.edu/>

²<http://wordnet.princeton.edu/man/wnstats.7WN>

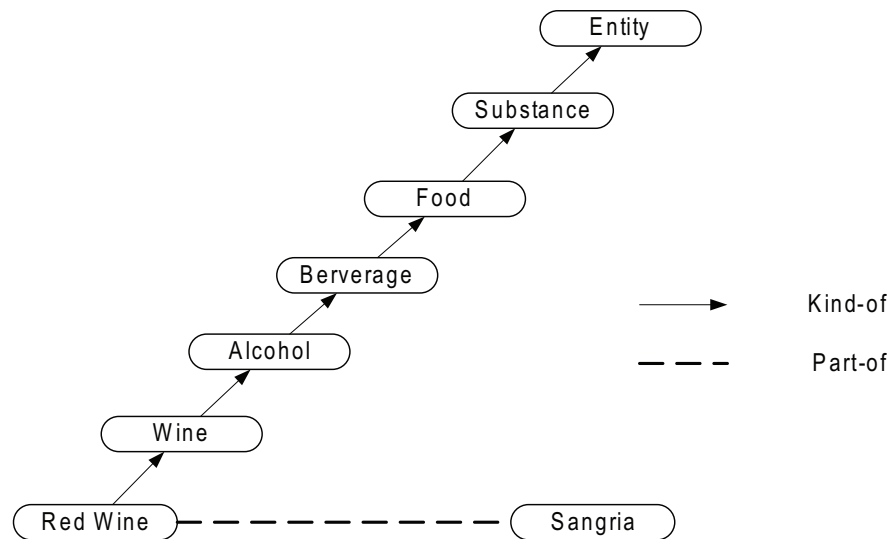


Figure D.1: An extract of WordNet for the synset *red-wine*.

The goal of WordNet is to develop a system that would be consistent with the knowledge acquired over the years about how human beings process language. In the case of hyponymy, psychological experiments³ revealed that individuals can access properties of nouns more quickly depending on when a characteristic becomes a defining property. That is, individuals can quickly verify that canaries can sing because a canary is a songbird (only one level of hyponymy), but require slightly more time to verify that canaries can fly (two levels of hyponymy) and even more time to verify canaries have skin (multiple levels of hyponymy). This suggests that we too store semantic information in a way that is much like WordNet, because we only retain the most specific information needed to differentiate one particular concept from similar concepts.

Ontology filtering assumes that user's preferences follows an ontology, and that such ontology can be used to infer missing preferences. From the inference mechanism, a similarity function has been derived that can compute the pairwise similarity between a pair of concepts in the ontology.

The similarity metric used by ontology filtering was tested on WordNet 2.0. Among the 155'000 words present, the experiment focused (only!) on the 117'097 nouns, which represents over 81'000 concepts⁴. Moreover, the meronym and holonym relations have been discarded as ontology filtering and other similarity metrics only use inheritance relation. Note however that these discarded relations count for less than 18% of all the relations in WordNet.

D.2 The GeneOntology

5

³<http://en.wikipedia.org/wiki/WordNet>

⁴<http://wordnet.princeton.edu/man2.0/wnstats.7WN>

⁵Due to author's lack of knowledge in biology, some of the content of this section has been extracted from the GeneOntology web site

The Gene Ontology⁶ (GO) project is a collaborative effort to address the need for consistent descriptions of gene products in different databases. The project began as a collaboration between three model organism databases, FlyBase⁷ (*Drosophila*), the *Saccharomyces* Genome Database⁸ (SGD) and the Mouse Genome Database⁹ (MGD), in 1998. Since then, the GO Consortium has grown to include many databases, including several of the world's major repositories for plant, animal and microbial genomes. See the GO Consortium page for a full list of member organizations.

The GO project has developed three structured controlled vocabularies (ontologies) that describe gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner. There are three separate aspects to this effort: first, the development and maintenance of the ontologies themselves; second, the annotation of gene products, which entails making associations between the ontologies and the genes and gene products in the collaborating databases; and third, development of tools that facilitate the creation, maintenance and use of ontologies.

The use of GO terms by collaborating databases facilitates uniform queries across them. The controlled vocabularies are structured so that they can be queried at different levels: for example, you can use GO to find all the gene products in the mouse genome that are involved in signal transduction, or you can zoom in on all the receptor tyrosine kinases. This structure also allows annotators to assign properties to genes or gene products at different levels, depending on the depth of knowledge about that entity.

Each entry in GO has a unique numerical identifier of the form GO:nnnnnnn, and a term name, e.g. cell, fibroblast growth factor receptor binding or signal transduction. Each term is also assigned to one of the three ontologies, molecular function, cellular component or biological process.

The majority of terms have a textual definition, with references stating the source of the definition. If any clarification of the definition or remarks about term usage are required, these are held in a separate comments field.

Many GO terms have synonyms; GO uses 'synonym' in a loose sense, as the names within the synonyms field may not mean exactly the same as the term they are attached to. Instead, a GO synonym may be broader or narrower than the term string; it may be a related phrase; it may be alternative wording, spelling or use a different system of nomenclature; or it may be a true synonym. This flexibility allows GO synonyms to serve as valuable search aids, as well as being useful for applications such as text mining and semantic matching. The relationship of the synonym to the term is recorded within the GO file.

The three organizing principles of GO are cellular component, biological process and molecular function. A gene product might be associated with or located in one or more cellular components; it is active in one or more biological processes, during which it performs one or more molecular functions. For example, the gene product cytochrome c can be described by the molecular function term oxidoreductase activity, the biological process terms oxidative phosphorylation and induction of cell death, and the cellular component terms mitochondrial matrix and mitochondrial inner membrane.

⁶<http://www.geneontology.org/>

⁷<http://flybase.bio.indiana.edu/>

⁸<http://www.yeastgenome.org/>

⁹<http://www.informatics.jax.org/>

D.2.1 Cellular component

A cellular component is just that, a component of a cell, but with the proviso that it is part of some larger object; this may be an anatomical structure (e.g. rough endoplasmic reticulum or nucleus) or a gene product group (e.g. ribosome, proteasome or a protein dimer). See the documentation on the cellular component ontology for more details.

D.2.2 Biological process

A biological process is series of events accomplished by one or more ordered assemblies of molecular functions. Examples of broad biological process terms are cellular physiological process or signal transduction. Examples of more specific terms are pyrimidine metabolism or alpha-glucoside transport. It can be difficult to distinguish between a biological process and a molecular function, but the general rule is that a process must have more than one distinct steps.

A biological process is not equivalent to a pathway; at present, GO does not try to represent the dynamics or dependencies that would be required to fully describe a pathway.

Further information can be found in the process ontology documentation.

D.2.3 Molecular function

Molecular function describes activities, such as catalytic or binding activities, that occur at the molecular level. GO molecular function terms represent activities rather than the entities (molecules or complexes) that perform the actions, and do not specify where or when, or in what context, the action takes place. Molecular functions generally correspond to activities that can be performed by individual gene products, but some activities are performed by assembled complexes of gene products. Examples of broad functional terms are catalytic activity, transporter activity, or binding; examples of narrower functional terms are adenylate cyclase activity or Toll receptor binding.

It is easy to confuse a gene product name with its molecular function, and for that reason many GO molecular functions are appended with the word "activity". The documentation on gene products explains this confusion in more depth. The documentation on the function ontology explains more about GO functions and the rules governing them.

D.2.4 Ontology structure

The terms in an ontology are linked by two relationships, *is_a* and *part_of*. *is_a* is a simple class-subclass relationship, where A *is_a* B means that A is a subclass of B; for example, nuclear chromosome *is_a* chromosome. *part_of* is slightly more complex; C *part_of* D means that whenever C is present, it is always a part of D, but C does not always have to be present. An example would be nucleus *part_of* cell; nuclei are always part of a cell, but not all cells have nuclei.

The ontologies are structured as directed acyclic graphs, which are similar to hierarchies but differ in that a child, or more specialized, term can have many parents, or less specialized, terms. For example, the biological process term hexose biosynthesis has two

parents, hexose metabolism and monosaccharide biosynthesis. This is because biosynthesis is a subtype of metabolism, and a hexose is a type of monosaccharide. When any gene involved in hexose biosynthesis is annotated to this term, it is automatically annotated to both hexose metabolism and monosaccharide biosynthesis, because every GO term must obey the true path rule: if the child term describes the gene product, then all its parent terms must also apply to that gene product.

Appendix E

Detailed experimental results

E.1 WordNet

The results in Table E.1 were computed using the following similarity metrics:

$$sim_{EDGE}(a, b) = \frac{(2 \times D) - len(a, b)}{\max D} \quad (E.1)$$

$$sim_{LEACOCK}(a, b) = -\log \left(\frac{len(a, b)}{2 \times D} \right) \quad (E.2)$$

$$sim_{RESNIK}(a, b) = \frac{\max_{c \in LCA(a, b)} IC(c)}{\max D} \quad (E.3)$$

$$sim_{LIN}(a, b) = \frac{2 \times IC(LCA(a, b))}{IC(a) + IC(b)} \quad (E.4)$$

$$sim_{JIANG}(a, b) = 1 - \frac{(IC(a) + IC(b) - 2 \times IC(LCA(a, b)))}{\max D} \quad (E.5)$$

$$sim_{OSS}(a, b) = \frac{\log(1 + 2\hat{\beta}(b, LCA(a, b))) - \log(\hat{\alpha}(a, LCA(a, b)))}{\max D} \quad (E.6)$$

where $\max D$ is the maximum value that is used to reduce the similarity values in the interval $[0, 1]$. The ontology which is being used is WordNet 2.0, and the human ratings are extracted from [Miller and Charles, 1991].

| Word Pair | | Human | Edge | Leacock | Resnik | Lin | Jiang | OSS |
|-------------|------------|-------|-------|---------|--------|------|-------|-------------|
| car | automobile | 0.98 | 1 | 1 | 1 | 1 | 1 | 1 |
| gem | jewel | 0.96 | 1 | 1 | 1 | 1 | 1 | 1 |
| journey | voyage | 0.96 | 0.97 | 1 | 0.66 | 0.84 | 0.88 | 0.96 |
| boy | lad | 0.94 | 0.97 | 1 | 0.76 | 0.89 | 0.88 | 0.95 |
| coast | shore | 0.93 | 0.97 | 1 | 0.78 | 0.98 | 0.99 | 0.98 |
| asylum | madhouse | 0.90 | 0.97 | 1 | 0.94 | 1 | 0.97 | 0.94 |
| magician | wizard | 0.88 | 1 | 1 | 1 | 1 | 1 | 1 |
| midday | noon | 0.86 | 1 | 1 | 1 | 1 | 1 | 1 |
| furnace | stove | 0.78 | 0.81 | 0.45 | 0.17 | 0.20 | 0.29 | 0.39 |
| food | fruit | 0.77 | 0.78 | 0.42 | 0.01 | 0.01 | 0.33 | 0.61 |
| bird | cock | 0.76 | 0.97 | 1 | 0.40 | 0.60 | 0.73 | 0.94 |
| bird | crane | 0.74 | 0.91 | 0.69 | 0.40 | 0.60 | 0.73 | 0.94 |
| tool | implement | 0.74 | 0.97 | 1 | 0.41 | 0.93 | 0.97 | 0.94 |
| brother | monk | 0.71 | 0.97 | 1 | 0.83 | 0.94 | 0.91 | 0.89 |
| crane | implement | 0.42 | 0.89 | 0.61 | 0.23 | 0.35 | 0.58 | 0.39 |
| lad | brother | 0.42 | 0.81 | 0.46 | 0.17 | 0.19 | 0.23 | 0.24 |
| journey | car | 0.29 | 0 | 0 | 0 | 0 | 0.17 | 0.32 |
| monk | oracle | 0.27 | 0.81 | 0.46 | 0.17 | 0.20 | 0.33 | 0.35 |
| cemetery | woodland | 0.24 | 0.75 | 0.39 | 0.01 | 0.01 | 0.15 | 0.10 |
| food | rooster | 0.22 | 0.64 | 0.28 | 0.01 | 0.01 | 0.36 | 0.61 |
| coast | hill | 0.21 | 0.89 | 0.61 | 0.50 | 0.59 | 0.61 | 0.22 |
| forest | graveyard | 0.21 | 0.75 | 0.39 | 0.01 | 0.01 | 0.15 | 0.21 |
| shore | woodland | 0.16 | 0.72 | 0.36 | 0.05 | 0.05 | 0.16 | 0.15 |
| monk | slave | 0.14 | 0.89 | 0.61 | 0.17 | 0.21 | 0.37 | 0.37 |
| coast | forest | 0.11 | 0.83 | 0.5 | 0.05 | 0.05 | 0.16 | 0.14 |
| lad | wizard | 0.11 | 0.87 | 0.56 | 0.17 | 0.18 | 0.20 | 0.21 |
| chord | smile | 0.03 | 0.72 | 0.36 | 0.24 | 0.27 | 0.34 | 0.36 |
| glass | magician | 0.03 | 0.694 | 0.33 | 0.16 | 0.17 | 0.21 | 0.11 |
| noon | string | 0.02 | 0 | 0 | 0 | 0 | 0.118 | 0.10 |
| rooster | voyage | 0.02 | 0 | 0 | 0 | 0 | 0.08 | 0.07 |
| Correlation | | 1 | 0.60 | 0.82 | 0.79 | 0.82 | 0.86 | 0.91 |

Table E.1: Correlation of various similarity metrics with human judgements collected by Miller and Charles.

E.2 Predefined ontology vs learnt ontologies

This section looks at the difference in recommendation accuracy for the ontology filtering that uses either a predefined ontology, or a set of learnt ontologies.

To test this aspect, the following experiment was performed. First, the MovieLens data set was split in two sets. The first one contains all the users who have rated less than 65 ratings and is used for generating the ontologies, while the other set is used for testing the recommendation strategy. Note that experiment setup is the same as in Section 8.2.1.

The baseline ontology is the one defined in Section 8.2. The learnt ontologies are obtained by applying Algorithm 7, which constructs a set of 15 different ontologies from the users who have rated less than 65 ratings. For this specific experiment, only the user's score is used for extracting the top-N items (i.e. the personalization coefficient ρ in Algorithm 6 is set to 1.0).

First, 5 items were inserted in the users' learning set, and ontology filtering was run on both sets of ontologies. Figure E.1 shows the accuracy of ontology filtering in the two situations. As it can be seen, the expert's ontology performs better when the learnt ontologies have either very few or a lot of leaf clusters. This shows that if the granularity (i.e.: the number of instances in each concept) of the ontology is inadequate, then recommendation accuracy decreases. To reduce this problem, ontology filtering also uses the popularity of concepts in the computation of the hybrid score (Algorithm 6).

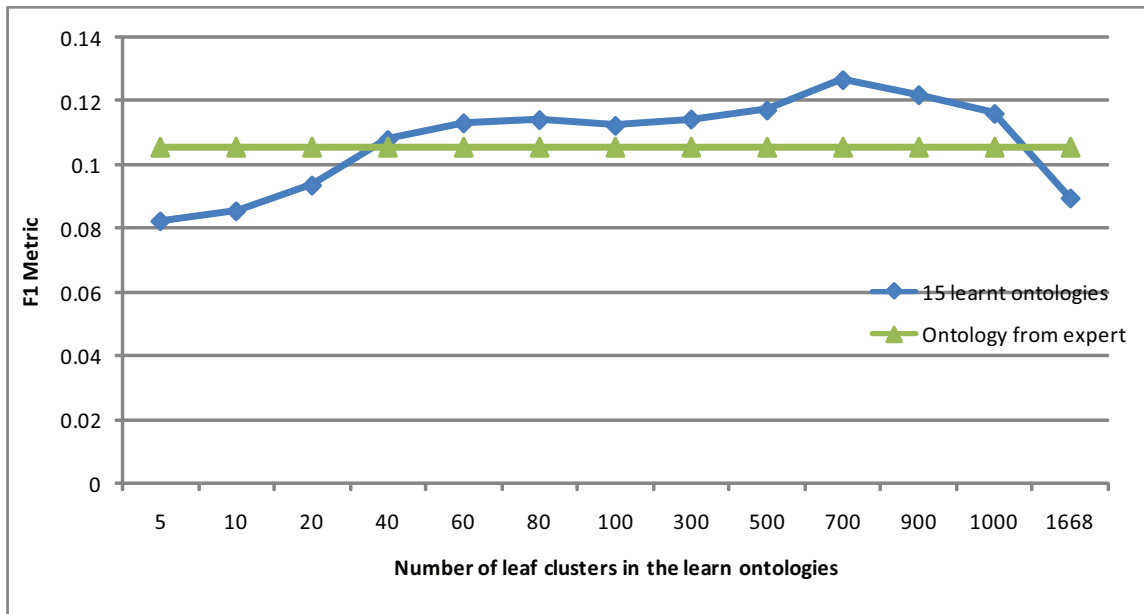


Figure E.1: Ontology filtering with 5 items in the users' learning set, and using either the ontology generated by the thesis's author, or the set of 15 ontologies learnt using Algorithm 7.

Second, the experiment was reproduced but using 50 items in the learning set. Surprisingly, ontology filtering with the learnt ontologies always perform better than with the unique expert's ontology. A potential reason for this is the fact that ontology filtering has

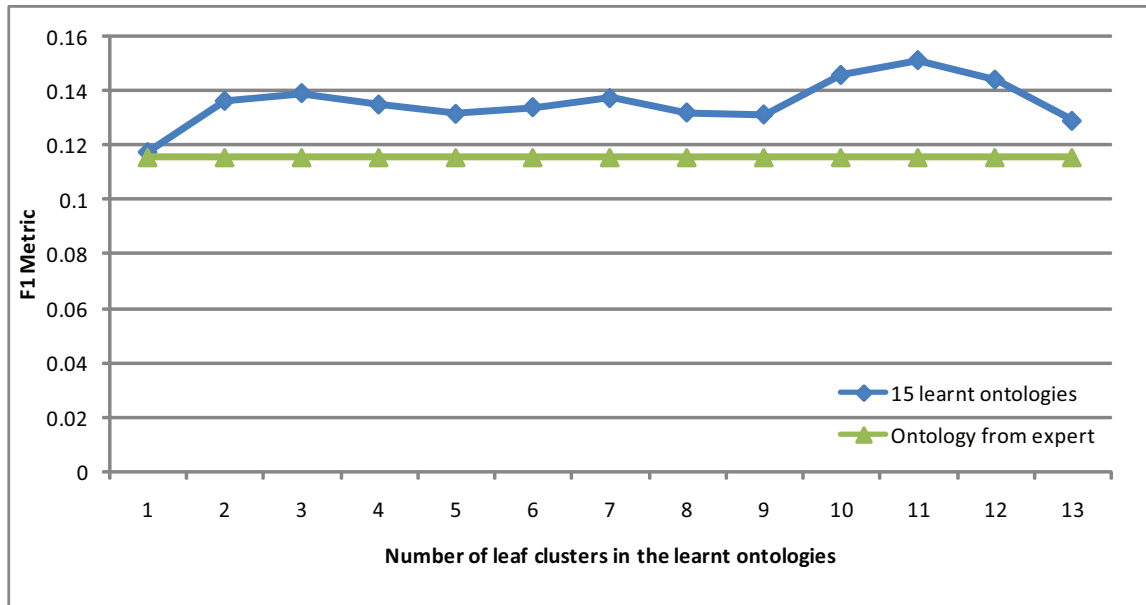


Figure E.2: Ontology filtering with 50 items in the users’ learning set and using either the ontology generated by the thesis’s author, or the set of 15 ontologies learnt using Algorithm 7.

15 ontologies to choose from, and contains enough ratings to select the best one. As a matter of fact, further experiments have shown that the performance of each learnt ontology taken separately is usually worse than the expert’s one.

It is acknowledged that there are too much bias in this experiment to conclude to any results. Moreover, and because no ontology could be designed for jokes, the experiment was only done on MovieLens and not on the Jester data set. Thus, this experiment is given only as pure information, and has not been used further.

The bias in the experiment could be reduced by asking a group of experts to also come up with 14 different ontologies. These ontologies, along with the one designed by the author, could then be used in order to assure that there are the same number of ontologies in both set-ups. This experiment is left as future work.

E.3 Improvement of OF over CF

This section gives the exact p-values for the improvement of ontology filtering over collaborative filtering. The details of the experiment can be found in Section 8.3.5.

| # clusters in ontology | <i>OF_5LS</i> | <i>OF_50LS</i> |
|------------------------|---------------|----------------|
| 10 | 2.15E-12 | 0.011 |
| 20 | 4.15E-12 | 0.316 |
| 40 | 3.66E-12 | 0.002 |
| 60 | 3.15E-12 | 0.008 |
| 80 | 2.64E-12 | 7.62E-06 |
| 100 | 4.38E-12 | 2.15E-12 |

Table E.2: P-values that measures the improvement of ontology filtering over collaborative filtering for the Jester data set.

| # clusters in ontology | <i>OF_5LS</i> | <i>OF_50LS</i> |
|------------------------|---------------|----------------|
| 10 | 4.42E-13 | 7.674E-12 |
| 20 | 1.91E-14 | 3.94E-12 |
| 40 | 5.69E-15 | 7.24E-13 |
| 60 | 6.30E-14 | 5.35E-11 |
| 80 | 4.41E-15 | 2.11E-12 |
| 100 | 8.02E-15 | 1.46E-08 |
| 250 | 0.033 | 0.036 |
| 500 | 5.92E-06 | 0.179 |
| 750 | 0.001 | 0.414 |
| 1000 | 0.001 | 0.102 |
| 1668 | 0.008 | 0.013 |

Table E.3: P-values that measures the improvement of ontology filtering over collaborative filtering for the MovieLens data set.

Bibliography

- [Adomavicius and Tuzhilin, 2005] Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749.
- [Agrawal et al., 1996] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. I. (1996). Fast discovery of association rules. pages 307–328.
- [Altschul, 1990] Altschul, S.F., G. W. M. W. M. E. L. D. (1990). Basic local alignment search tool. In *Jour. of molecular biology*, volume 215, pages 403–410.
- [Anderson, 1983] Anderson, J. R. (1983). A spreading activation theory of memory. In *A spreading activation theory of memory*, volume 22, pages 261–295.
- [Balabanovic and Shoham, 1997] Balabanovic, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72.
- [Belkin and Croft, 1992] Belkin, N. J. and Croft, W. B. (1992). Information filtering and information retrieval: two sides of the same coin? *Communications of the ACM*, 35(12):29–38.
- [Bender et al., 2005] Bender, M. A., Farach-Colton, M., Pemmasani, G., Skiena, S., and Sumazin, P. (2005). Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75 – 94.
- [Bergman,] Bergman, M. K. The deep web: Surfacing hidden value. *Journal of Electronic Publishing*.
- [Bhaumik et al., 2006] Bhaumik, R., Williams, C., Mobasher, B., and Burke, R. (2006). Securing collaborative filtering against malicious attacks through anomaly detection. In *Proceedings of the 4th Workshop on Intelligent Techniques for Web Personalization (ITWP'06), at AAAI 2006*.
- [Billsus and Pazzani, 1998] Billsus, D. and Pazzani, M. J. (1998). Learning collaborative information filters. In *Proc. 15th International Conference on Machine Learning*, pages 46–54. Morgan Kaufmann, San Francisco, CA.
- [Blythe, 2002] Blythe, J. (2002). Visual exploration and incremental utility elicitation. In *18th National Conference on Artificial Intelligence*, pages 526 – 532.

- [Bradley et al., 2000] Bradley, K., Rafter, R., and Smyth, B. (2000). *Case-Based User Profiling for Content Personalization (AH2000)*, pages 133–143. Springer-Verlag.
- [Brand, 2003] Brand, M. (2003). Fast online svd revisions for lightweight recommender systems. In *3rd SIAM International Conference on Data Mining*.
- [Breese et al., 1998] Breese, J., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *14th Conf. on Uncertainty in Artificial Intelligence*, pages 43–52.
- [Burke, 2000] Burke, R. (2000). Knowledge-based recommender systems. In *Encyclopedia of Library and Information Systems. Vol. 69, Supplement 32*. New York: Marcel Dekker.
- [Burke, 2002] Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331 – 370.
- [Burke et al., 1997] Burke, R. D., Hammond, K. J., and Young, B. C. (1997). The findme approach to assisted browsing. *IEEE Expert: Intelligent Systems and Their Applications*, 12(4):32–40.
- [Burke et al., 2006] Burke, R. D., Mobasher, B., Williams, C., and Bhaumik, R. (2006). Detecting profile injection attacks in collaborative recommender systems. *cec-eee*, 0:23.
- [Canny, 2002] Canny, J. (2002). Collaborative filtering with privacy. In *IEEE Symposium on Security and Privacy, 2002*, pages 45– 57.
- [Castagnos and Boyer, 2006] Castagnos, S. and Boyer, A. (2006). A client/server user-based collaborative filtering algorithm: Model and implementation. In *17th European Conference on Artificial Intelligence (ECAI 2006)*, pages 617–621.
- [Cho et al., 2005] Cho, Y. H., Kim, J. K., and Ahn, D. H. (2005). *A Personalized Product Recommender for Web Retailers*, volume LNCS 3398, pages 206–305. Springer-Verlag.
- [Claypool et al., 1999] Claypool, M., and Makoto Wased, P. L., and Brown, D. (1999). Implicit interest indicators. In *ACM SIGIR Workshop on Recommender Systems*.
- [Claypool et al., 2001] Claypool, M., Gokhale, A., and Miranda, T. (2001). Combining content-based and collaborative filters in an online newspaper. In *IUI'01*, pages 33–40.
- [Clerkin et al., 2001] Clerkin, P., Cunningham, P., and Hayes, C. (2001). Ontology discovery for the semantic web using hierarchical clustering. In *Semantic Web Mining Workshop*.
- [Connor and Herlocker, 2001] Connor, M. and Herlocker, J. (2001). Clustering items for collaborative filtering. In *SIGIR-2001 Workshop on Recommender Systems*.
- [Cost and Salzberg, 1993] Cost, S. and Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–78.

- [Davies and Weeks, 2004] Davies, J. and Weeks, R. (2004). Quizrdf: Search technology for the semantic web. In *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 4*, page 40112, Washington, DC, USA. IEEE Computer Society.
- [Dempster et al., 1977] Dempster, A., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the *em* algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.
- [Ding et al., 2001] Ding, C., He, X., Zha, H., Gu, M., and Simon, H. (2001). Spectral min-max cut for graph partitioning and data clustering. *Technical Report TR-2001-XX, Lawrence Berkely National Laboratory, University of California*.
- [Duda and Hart, 1973] Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis*. A Wiley-Interscience Publication, New York: Wiley.
- [Fisher, 1987] Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning* 2, (2):139–172.
- [Frakes and Baeza-Yates, 1992] Frakes, W. and Baeza-Yates, R. (1992). *Information Retrieval: Data Structures & Algorithms*.
- [Getoor and Sahami,] Getoor, L. and Sahami, M. Using probabilistic relational models for collaborative filtering. In *WebKDD'99*.
- [GO, 2000] GO, T. G. O. C. (2000). Gene ontology: tool for the unification of biology. In America, N., editor, *Nature Genetic*, volume 25, pages 25–29.
- [Goldberg et al., 1992] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70.
- [Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontologies. *Knowledge acquisition*, 5(2):199–220.
- [Guha et al., 2003] Guha, R., McCool, R., and Miller, E. (2003). Semantic search. In *12th International World Wide Web Conference, WWW'03*, pages 700 – 709.
- [Ha and Haddawy, 1997] Ha, V. and Haddawy, P. (1997). Problem-focused incremental elicitation of multi-attribute utility model. In *13th Conference on Uncertainty in Artificial Intelligence, UAI'97*, pages 215 – 222.
- [Ha and Haddawy, 1999] Ha, V. and Haddawy, P. (1999). A hybrid approach to reasoning with partially elicited preference models. In *15th Conference on Uncertainty in Artificial Intelligence, UAI'99*, pages 263 – 270.
- [Hawking, 2006] Hawking, D. (2006). Web search engines: Part 1. *IEEE Computer*, 39(6):86–88.

- [Herlocker et al., 1999] Herlocker, J., Borchers, A., Konstan, J., and Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR'1999*, pages 230–235.
- [Herlocker et al., 2004] Herlocker, J., Konstan, J., Terveen, L., and Riedl, J. (2004). Evaluating collaborative filtering recommender systems. *acm transaction on information systems*. In *ACM Transactions on Information Systems*, volume 22, pages 5–53.
- [Hill and Terveen, 1996] Hill, W. and Terveen, L. (1996). Using frequency-of-mention in public conversations for social filtering. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 106–112, New York, NY, USA. ACM Press.
- [Hofmann, 1999] Hofmann, T. (1999). Probabilistic latent semantic analysis. In *15th Conference on Uncertainty in Artificial Intelligence, UAI'99*, Stockholm.
- [Hu et al., 2006] Hu, N., Pavlou, P., and Zhang, J. (2006). Can Online Reviews Reveal a Product's True Quality? In *Proceedings of ACM Conference on Electronic Commerce, EC'06*, pages 324 – 330.
- [Ittner et al., 1995] Ittner, D. J., Lewis, D. D., and Ahn, D. D. (1995). Text categorization of low quality images. In *SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval*, pages 301–315.
- [Janecek et al., 2005] Janecek, P., Schickel-Zuber, V., and Pu, P. (2005). *Concept Expansion Using Semantic Fisheye Views*, volume LNCS 3815, pages 273–282. Springer-Verlag.
- [Jiang and Conrath, 1997] Jiang, J. J. and Conrath, D. W. (1997). Semantic Similarity based on corpus Statistics and lexical taxonomy. In *10th International Conference Research on Computational Linguistics, ROCLING X'1997*.
- [Keeney and Raiffa, 1993] Keeney, R. and Raiffa, H. (1993). *Decisions with Multiple Objectives: Preference and Value Tradeoffs*. Cambridge University Press.
- [Kohrs and Merialdo,] Kohrs, A. and Merialdo, B. Clustering for collaborative filtering applications. In *Computational Intelligence for Modelling, Control, CIMCA'99*.
- [Konstan et al., 1997] Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. (1997). GroupLens: Applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87.
- [Lang, 1995] Lang, K. (1995). NewsWeeder: learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.
- [Leacock, 1997] Leacock, C., C. M. (1997). Combining local context and WordNet similarity for word sense identification. In *Fellbaum*, pages 265 – 283.

- [Li et al., 2005] Li, Q., Kin, B. M., Myaeng, S., and H. (2005). Clustering for probabilistic model estimation for cf. In *Proc. of 14th International World Wide Web Conference, WWW'05*, pages 1104 – 1005.
- [Lim and Teh, 2007] Lim, Y. J. and Teh, Y. W. (2007). Variational bayesian approach to movie rating prediction. In *KDD Cup and Workshop 2007*, pages 15–21.
- [Lin and Chen, 2005] Lin, C.-R. and Chen, M.-S. (2005). Combining partitional and hierarchical algorithms for robust and efficient data clustering with cohesion self-merging. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):145–159.
- [Lin, 1998] Lin, D. (1998). An information-theoretic definition of similarity. In *Proceedings of the 15th International Conference on Machine Learning*, pages 296 – 304.
- [Linden et al., 2003] Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations item-to-item collaborative filtering. *IEEE Internet Computing*, pages 76 – 80.
- [Liu and Lieberman, 2002] Liu, H. and Lieberman, H. (2002). Robust Photo Retrieval Using World Semantics. In *LREC 2002 - Workshop on Creating and Using Semantics for Information Retrieval and Filtering*, pages 15–20.
- [Liu and Maes, 2005] Liu, H. and Maes, P. (2005). InterestMap: Harvesting social network profiles for recommendations. In *IUI'05: Workshop on the Next Stage of Recommender Systems Research*, pages 54–59.
- [Lord et al., 2003] Lord, P., Stevens, R., Brass, A., and Goble, C. (2003). Semantic Similarity Measures as Tools For Exploring The Gene Ontology. In *Pacific Symposium on Biocomputing*, volume 8, pages 601 – 612.
- [Lyman et al., 2003] Lyman, P., Varian, H. R., Charles, P., Good, N., Jordan, L. L., and Pal, J. (2003). How much information 2003? *Journal of Electronic Publishing*.
- [Maguitman, 2005] Maguitman, A., M. F. R. H. V. A. (2005). Algorithmic Detection of Semantic Similarity. In *Proceedings of the International World Wide Web Conference, WWW'05*, pages 107 – 116.
- [McGinty and Smyth, 2002] McGinty, L. and Smyth, B. (2002). Evaluating preference-based feedback in recommender systems. *LNAI 2462*:209–214.
- [McGuinness, 2002] McGuinness, D. (2002). A constant time collaborative filtering algorithm. In *Spinning the semantic web: bringing the world wide web to its full potential*, pages 171–196. MIT press.
- [McLaughlin and Herlocker, 2004] McLaughlin, M. R. and Herlocker, J. L. (2004). A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *27th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR'04*, pages 329–336.

- [Melville et al., 2001] Melville, P., Mooney, R., and Nagarajan, R. (2001). Content-boosted collaborative filtering. In *ACM SIGIR Workshop on Recommender Systems*.
- [Middleton et al., 2004] Middleton, S. E., Shadbolt, N. R., and Roure, D. C. D. (2004). Ontological user profiling in recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):54–88.
- [Miller et al., 1993] Miller, G., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1993). Introduction to WordNet: An On-line Lexical Database. Technical report, Cognitive Science Laboratory, Princeton University.
- [Miller and Charles, 1991] Miller, G. A. and Charles, W. G. (1991). Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28.
- [Mitchell, 1997] Mitchell, T. (1997). *Machine Learning*. McGraw-Hill, New-York.
- [Mobasher et al., 2004] Mobasher, B., Jin, X., and Zhou, Y. (2004). Semantically enhanced collaborative filtering on the web. In *Proceedings of EWMF03*, volume LNAI 3209, pages 57 – 76. Springer-Verlag.
- [Mooney et al., 1998] Mooney, R., Bennett, P., and Roy, L. (1998). Book recommending using text categorization with extracted information. In *Proceedings of the AAAI Workshop on Recommender Systems*, pages 70–74.
- [Mooney and Roy, 2000] Mooney, R. J. and Roy, L. (2000). Content-based book recommending using learning for text categorization. In *Proceedings of DL-00, 5th ACM Conference on Digital Libraries*, pages 195–204, San Antonio, US. ACM Press, New York, US.
- [Neumann and Morgenstern, 1944] Neumann, J. V. and Morgenstern, O. (1944). *The Theory of Games and Economic Behavior*. Princeton University Press.
- [Olston and Chi, 2003] Olston, C. and Chi, E. H. (2003). Scenttrails: Integrating browsing and searching on the web. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 10(3):177–197.
- [O’Mahony et al., 2004] O’Mahony, M., Hurley, N., Kushmerick, N., and Silvestre, G. (2004). Collaborative recommendation: A robustness analysis. *ACM Transactions on Internet Technology (TOIT)*, 4(4):344–377.
- [O’Sullivan et al., 2004] O’Sullivan, D., Smyth, B., Wilson, D., McDonald, K., and Smeaton, A. (2004). Improving the quality of the personalized electronic program guide. *User Modeling and User-Adapted Interaction*, 14(1):5 – 36.
- [O’Sullivan et al., 2003] O’Sullivan, D., Smyth, B., and Wilson, D. C. (2003). Explicit vs implicit profiling - a case-study in electronic programme guides. In *IJCAI*, pages 1351 – 1353.

- [Park et al., 2006] Park, S.-T., Pennock, D., Madani, O., Good, N., and DeCoste, D. (2006). Naive filterbots for robust cold-start recommendations. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 699–705, New York, NY, USA. ACM Press.
- [Parsons et al., 2004] Parsons, J., Ralph, P., and Gallagher, K. (2004). Using viewing time to infer user preference in recommender systems. In *AAAI Workshop in Semantic Web Personalization*, pages 57 – 76.
- [Pavlov and Pennock, 2002] Pavlov, D. Y. and Pennock, D. M. (2002). A maximum entropy approach to collaborative filtering in dynamic, sparse, high-dimensional domains. In *Conference on Neural Information Processing Systems*, pages 175–186.
- [Payne et al., 1988] Payne, J. W., Bettman, J., and Johnson, E. (1988). Adaptive strategy selection in decision making. *Experimental Psychology: Learning, Memory, and Cognition*, 14(3):534–552.
- [Pazzani and Billsus, 1997] Pazzani, M. J. and Billsus, D. (1997). Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331.
- [Polat and Du, 2003] Polat, H. and Du, W. (2003). Privacy-preserving collaborative filtering using randomized perturbation techniques. In *The Third IEEE International Conference on Data Mining (ICDM'03)*, page 625.
- [Quinlan, 1990] Quinlan, J. R. (1990). Induction of decision trees. In Shavlik, J. W. and Dietterich, T. G., editors, *Readings in Machine Learning*. Morgan Kaufmann. Originally published in *Machine Learning* 1:81–106, 1986.
- [Reilly et al., 2004] Reilly, J., McCarthy, K., McGinty, L., and Smyth, B. (2004). Dynamic critiquing. In *Proceedings of the Seventh European Conference on Case-Based Reasoning (ECCBR-04)*, pages 767–777.
- [Resnick et al., 1994] Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P., and Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina. ACM.
- [Resnick and Varian, 1997] Resnick, P. and Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, 40(3):56–58.
- [Resnik, 1995] Resnik, P. (1995). Using information content to evaluate semantic similarity. In *Proceedings of the IJCAI'05*, pages 448 – 453.
- [Rissland, 2006] Rissland, E. (2006). Ai and similarity. *IEEE Intelligent Systems*, 21(3):39–49.

- [Salakhutdinov et al., 2007] Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 791–798, New York, NY, USA. ACM Press.
- [Sarwar et al., 2000a] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2000a). Analysis of recommendation algorithms for e-commerce. In *ACM Conference on Electronic Commerce*, pages 158–167.
- [Sarwar et al., 2000b] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2000b). Application of dimensionality reduction in recommender systems—a case study. In *ACM WebKDD Workshop, WebKDD'2000*.
- [Sarwar et al., 2001] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *10th International World Wide Web Conference, WWW'01*, pages 285–295.
- [Sarwar et al., 2002] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2002). Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Fifth International Conference on Computer and Information Technology*.
- [Sarwar et al., 1998] Sarwar, B., Konstan, J., Borchers, A., Herlocker, J., Miller, B., and Riedl, J. (1998). Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Computer Supported Cooperative Work*, pages 345–354.
- [Schaffer et al., 1999] Schaffer, J. B., Konstan, J., and Riedl, J. (1999). Recommender systems in e-commerce. In *1st ACM conference on Electronic commerce, E-Commerce 99*, pages 158–166.
- [Schein et al., 2002] Schein, A., Popescu, A., Ungar, L., and Pennock, D. (2002). Methods and metrics for cold-start recommendations. In *25th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR'02*, pages 263–260.
- [Seo and Ozden, 2004] Seo, C. and Ozden, B. (2004). Ontology-based file naming through hierarchical conceptual clustering. In *Technical Report, University of Southern California*.
- [Sieg et al., 2005] Sieg, A., Mobasher, B., R. Burke, a. G. P., and Lytinen, S. (2005). Representing user information context with ontologies. In *Proceedings of HCI International 2005 Conference*, pages 158–167.
- [Simazu et al., 2001] Simazu, H., Shibata, A., and Nihei, K. (2001). Expertguide: A conversational case-based reasoning tool for developing mentors in knowledge spaces. *Applied Intelligence*, 14(1):33–48.

- [Smyth and McClave, 2001] Smyth, B. and McClave, P. (2001). Similarity vs. diversity. In *ICCBR '01: Proceedings of the 4th International Conference on Case-Based Reasoning*, pages 347–361, London, UK. Springer-Verlag.
- [Smyth and McGinty, 2003] Smyth, B. and McGinty, L. (2003). An analysis of feedback strategies in conversational recommender systems. In *14th National Conference on Artificial and Cognitive Science (AICS-2003)*.
- [Srebro and Jaakkola, 2003] Srebro, N. and Jaakkola, T. (2003). Weighted low rank approximation. In *20th International Conference on Machine Learning*.
- [Stolze, 2000] Stolze, M. (2000). Soft navigation in electronic product catalogs. *Journal on Digital Libraries*, 3(1):60–66.
- [Takacs et al., 2007] Takacs, G., Pillaszy, I., Nemeth, B., and Tikk, D. (2007). On the gravity recommendation system. In *KDD Cup and Workshop 2007*, pages 22–29.
- [Terveen and Hill, 2001] Terveen, L. and Hill, W. (2001). *Human-Computer Collaboration in Recommender Systems*. Addison Wesley.
- [Ungar and Foster, 1998] Ungar, L. and Foster, D. (1998). Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems*. AAAI Press, Menlo Park California.
- [Viappiani et al., 2006] Viappiani, P., Faltings, B., and Pu, P. (2006). Preference-based search using example-critiquing with suggestions. *JAIR*, 27:465–503.
- [Viappiani et al., 2002] Viappiani, P., Pu, P., and Faltings, B. (2002). Acquiring user preferences for personal agents. In *AAAI Fall Symposium*, North Falmouth, MA. AAAI Press.
- [Walley, 1996] Walley, P. (1996). Measures of uncertainty in expert systems. *Journal of Artificial Intelligence*, 83:1 – 58.
- [Widrow and Hoff, 1988] Widrow, B. and Hoff, M. E. (1988). Adaptive switching circuits. *Neurocomputing: foundations of research*, pages 123–134.
- [Winterfeld and Edwards, 1986] Winterfeld, D. and Edwards, W. (1986). *Decision Analysis and Behavioral Research*. Cambridge University Press; 1 edition.
- [Zhang and Pu, 2004] Zhang, J. and Pu, P. (2004). Survey of solving multi-attribute decision problems. Technical Report No. IC/2004/54 200454, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland).
- [Zhang and Pu, 2005] Zhang, J. and Pu, P. (2005). Effort and accuracy analysis of choice strategies for electronic product catalogs. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 808–814, New York, NY, USA. ACM Press.
- [Zhao and Karypis, 2005] Zhao, Y. and Karypis, G. (2005). Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 10:141–168.

- [Zhuang, 2001] Zhuang, J. Y. L. W. H. Z. Y. (2001). Thesaurus-aided approach for image browsing and retrievals. In *IEEE International Conference on Multimedia and Expo. ICME 2001.*, pages 1135–1138.
- [Ziegler et al., 2004] Ziegler, C.-N., Lausen, G., and Schmidt-Thieme, L. (2004). Taxonomy-driven computation of product recommendations. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 406–415, New York, NY, USA. ACM Press.
- [Ziegler et al., 2005] Ziegler, C.-N., McNee, S., Konstan, J., and Lausen, G. (2005). Improving Recommendation List Through Topic Diversification. In *14th International World Wide Web Conference, WWW'05*, pages 22 – 32.

Appendix F

Curriculum Vitae

Personal Information

Name: Vincent Jean Fabrice SCHICKEL-ZUBER.

Date and Place of Birth: January 8th 1979 in Nantes, France.

Nationality and Status: French, single.

Languages: French and English.

Address: Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
Artificial Intelligence Laboratory
Office IN R 230 - Station 14
CH - 1015 Lausanne.

Contact: *Phone* +41 21 693 6738
Fax +41 21 693 5225
Email vincent.schickel-zuber@epfl.ch.

Education

| | |
|----------------|--|
| 2004 - Present | Swiss Federal Institute of Technology in Lausanne (EPFL) Ph.D. Computer Science, Thesis: <i>Ontology Filtering</i> . |
| 2003 - 2004 | Swiss Federal Institute of Technology in Lausanne (EPFL) Postgraduate course Computer Science, GPA: 94%. |
| 1997 - 2003 | Swiss Federal Institute of Technology in Lausanne (EPFL) MSc Computer Science, GPA: 94%. |
| 1993 - 1997 | Edgehill College, Bideford, England High School , GPA: 80%. |

Professional Experience

| | |
|----------------|--|
| 2005 - Present | Swiss Federal Institute of Technology in Lausanne (EPFL) IT Administrator and Teaching Assistant at the AI Lab. |
| 2005 - Present | Computer Science Alumni Association (EPFL) IT Administrator and Comite member . |
| 2003 - 2004 | Swiss Federal Institute of Technology in Lausanne (EPFL) Research Assistant and Teaching Assistant in C++. |
| 2004 - 2004 | Web programmer Freelance Php and MySQL Technology. |
| 2003 - 2004 | Nokia Research in Boston - USA Intern at the Agent Technology Group (ATG). |

Publications

Journal papers

Recommender Systems using Ontology Filtering.
Vincent Schickel-Zuber and Boi Faltings
Journal of Artificial Intelligence Research, reviewing, 2007.

Patents

Methods of Inferring User Preferences Using Ontologies.
Vincent Schickel-Zuber and Boi Faltings
US Provisional Patent number 60819,290, 2006.
US Patent Pending, 2007.

Conference and workshop papers

Using Hierarchical Clustering for Learning the Ontologies used in Recommendation Systems, Vincent Schickel-Zuber and Boi Faltings. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*, pages 599 – 608, San Jose, USA, August 12–15, 2007.

OSS: A semantic Similarity Function based on Hierarchical Ontologies, Vincent Schickel-Zuber and Boi Faltings. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 551 – 556, Hyderabad, India, January 6–12, 2007.

Using an Ontological A-priori Score To Infer User's Preferences, Vincent Schickel-Zuber and Boi Faltings. In *Proceedings of Recommender Systems Workshop part of the 17th European Conference on Artificial Intelligence (ECAI'06)*, pages 1023 – 106, Riva del Garda, Italy, August 28–29, 2006.

Inferring User's Preferences using Ontologies, Vincent Schickel-Zuber and Boi Faltings. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, pages 1413 – 1418, Boston, USA, July 16–20, 2006.

Overcoming Incomplete User Models in Recommendation Systems via an Ontology, Vincent Schickel-Zuber and Boi Faltings. In *Lecture Notes in Computer Science (LNCS 4198)*, Volume 4198, pages 39 – 57, Springer-Verlag, 2006.

Heterogeneous Attribute Utility Model: A new approach for modeling user profiles for recommendation systems, Vincent Schickel-Zuber and Boi Faltings. In *Proceedings of Workshop on Knowledge Discovery in the web part of the 21st International Conference on Knowledge Discovery and Data Mining (KDD'05)*, pages 80 – 91, Chicago, USA, August 21, 2005.

Co-authored papers

Stimulating Preference Expression Using Suggestions, Paolo Viappiani, Boi Faltings, Vincent Schickel-Zuber and Pearl Pu. In *Proceedings of 2005 AAAI Fall Symposium on Mixed-Initiative Problem Solving Assistants (AAAI'05)*, pages 128 – 133, 2005.

Concept Expansion Using semantic Fisheye Views, Paul Janecek, Vincent Schickel-Zuber and Pearl Pu. In *Proceedings of the 8th International Conference on Asian Digital Libraries (ICADL'05)*, pages 273 – 282, Bangkok , Thailand, Dec 12- 15, 2005 .

Concept Expansion Using semantic Fisheye Views, Paul Janecek, Vincent Schickel-Zuber and Pearl Pu. In *Lecture Notes in Computer Science (LNCS 3815)* , Volume 3815, Springer-Verlag, 2005 .

Stimulating Preference Expression Using Suggestions, Paolo Viappiani, Boi Faltings, Vincent Schickel-Zuber and Pearl Pu. In *Proceedings of the Multidisciplinary Workshop on Advances in Preference (IJCAI'05)*, pages 186 – 191, Edinburgh, Scotland , July 31 - August 1, 2005 .

Specification of a methodology for syntactic and semantic versioning (D231) , Wolf Winkler, Max Vlkel, York Sure, Vincent Schickel-Zuber, Walter Binder, Vassilis Tzouvaras, Diego Ponte, Chiara Zini, Matteo Bonifacio, Sebastian Ryszard Kruk, Marcin Synak . In *Deliverable 231, Knowledge Web*, 2004 .

