On-demand Feature Recommendations Derived from Mining Public Product Descriptions

Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang,
Bamshad Mobasher, Carlos Castro-Herrera, Mehdi Mirakhorli
DePaul University
243 S. Wabash Ave, Chicago, IL 60604
{dumitru.horatiu, mgibiec, negar.hariri}@gmail.com
{jhuang, mobasher, ccastroh}@cs.depaul.edu, m.mirakhorli@acm.org

ABSTRACT

We present a recommender system that models and recommends product features for a given domain. Our approach mines product descriptions from publicly available online specifications, utilizes text mining and a novel incremental diffusive clustering algorithm to discover domain-specific features, generates a probabilistic feature model that represents commonalities, variants, and cross-category features, and then uses association rule mining and the *k*-Nearest-Neighbor machine learning strategy to generate product specific feature recommendations. Our recommender system supports the relatively labor-intensive task of domain analysis, potentially increasing opportunities for re-use, reducing time-to-market, and delivering more competitive software products. The approach is empirically validated against 20 different product categories using thousands of product descriptions mined from a repository of free software applications.

Categories and Subject Descriptors

D.2.13 [**Reusable Software**]: Domain Engineering; H.3.3 [**Information Storage and retrieval**]: Retrieval models, Clustering, Query formulation

General Terms

Algorithms, Documentation

Keywords

Domain analysis, Recommender Systems, Clustering

1. INTRODUCTION

Domain analysis is the process of analyzing related software systems to identify, organize, and represent features common to systems within a domain [4, 17]. Although domain analysis is typically performed as part of the product line development process, its use can also be beneficial in single-application projects. For example, an analyst eliciting requirements for a new Anti-Virus software system might evaluate the features provided by competing products [18], or a developer might analyze an existing health-care administration system to extract a set of re-usable assets for use in a new product. In both of these cases, the identification and timely reuse of domain assets could potentially reduce development costs, shorten time-to-market, improve quality, and increase product competitiveness.

Although several domain analysis techniques, such as Feature Oriented Domain Analysis (FODA) [17], and the Domain Analysis and Reuse Environment (DARE) [13] have been developed, these approaches generally assume that analysts utilize existing requirements documentation and then manually or semi-manually evaluate the documentation to extract features. They provide little upfront support for automating the task of identifying features or for understanding their composition rules. As a result, domain analysis tends to be quite labor intensive, dependent upon the expertise of available analysts or subject matter experts, and constrained by the availability of requirements specifications from previous related projects.

To address these problems, researchers have applied data-mining and natural language processing (NLP) techniques to automate the process of mining features from requirements specifications or project repositories [3, 8, 25]. However, these techniques are only useful when an organization has an available repository of requirements specifications or other related assets. Furthermore, the automated analysis of existing documents is constrained to the features described in those documents. This approach therefore fails to explore the full range of features that might be offered by competitors and does not help analysts to explore ideas beyond the confines of their own existing products.

In this paper, we address these limitations through presenting a novel approach for automating the feature analysis process. In contrast to previous methods that extract features from proprietary project repositories, our approach mines raw feature descriptions, referred to from now on as *descriptors*, from thousands of product specifications found on Internet sites which host or market software packages. It then uses a novel clustering algorithm, especially suited for this task, to automatically discover and characterize product features. It draws inferences about the relationships between features that are not possible when features are extracted from only a handful of requirements specifications.

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

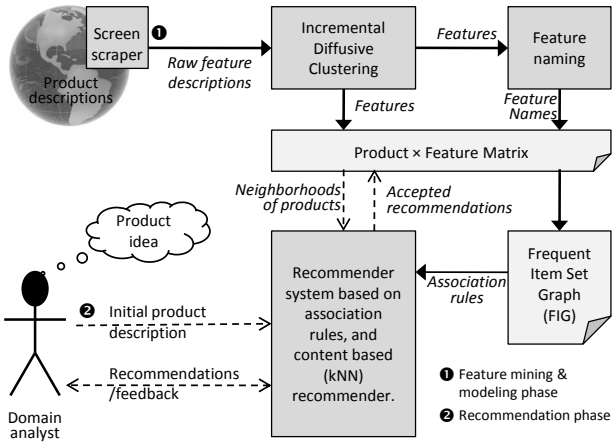


Figure 1: Feature Extraction and Recommendations

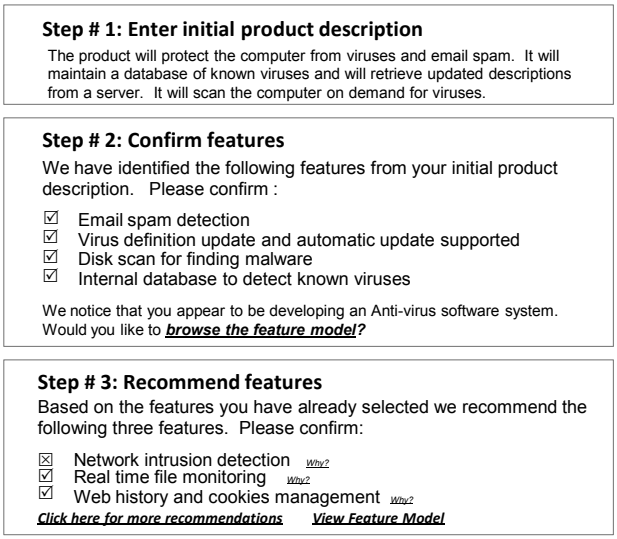


Figure 2: Example of Feature Recommendations

Specifically, we use association rule mining [1, 2] to identify affinities among product features, and we employ kNN (k-Nearest-Neighbor) machine learning strategy to identify similar products and make predictions about the existence of features not previously identified. As a result, our approach not only generates a feature model for a given product category, but also introduces a sophisticated recommender system, capable of recommending features, and combinations of features, which can provide useful insights during the domain analysis process.

2. OVERVIEW

Our Feature Recommender System includes several different components as illustrated in Figure 1. The first component is the *Screen scraper*, responsible for mining *descriptors* from online product specifications. The second component is the *incremental diffusive clustering* algorithm, which clusters descriptors into *features*. This clustering algorithm, which is capable of identifying both dominant and latent themes from descriptors has been shown in our prior work to perform well for the task of clustering requirements [14]. The third component is a product-by-feature matrix which is generated as a by-product of the clustering process and is then used to construct a *frequent itemset graph* (FIG) [23, 29] from which feature *association rules* can be generated. The final component is the recommender system itself. This system utilizes both the product-by-feature matrix and the FIG to generate on-demand feature recommendations.

The feature recommendation process is initiated when a user provides a preliminary product description. This description is automatically parsed and analyzed to create a partial *product profile*. This profile may be relatively sparse, representing the case where a user is quite unfamiliar with the domain; or it may be relatively rich, representing the case where a user understands the core features of the domain, but is searching for additional feature ideas that might competitively differentiate the targeted product.

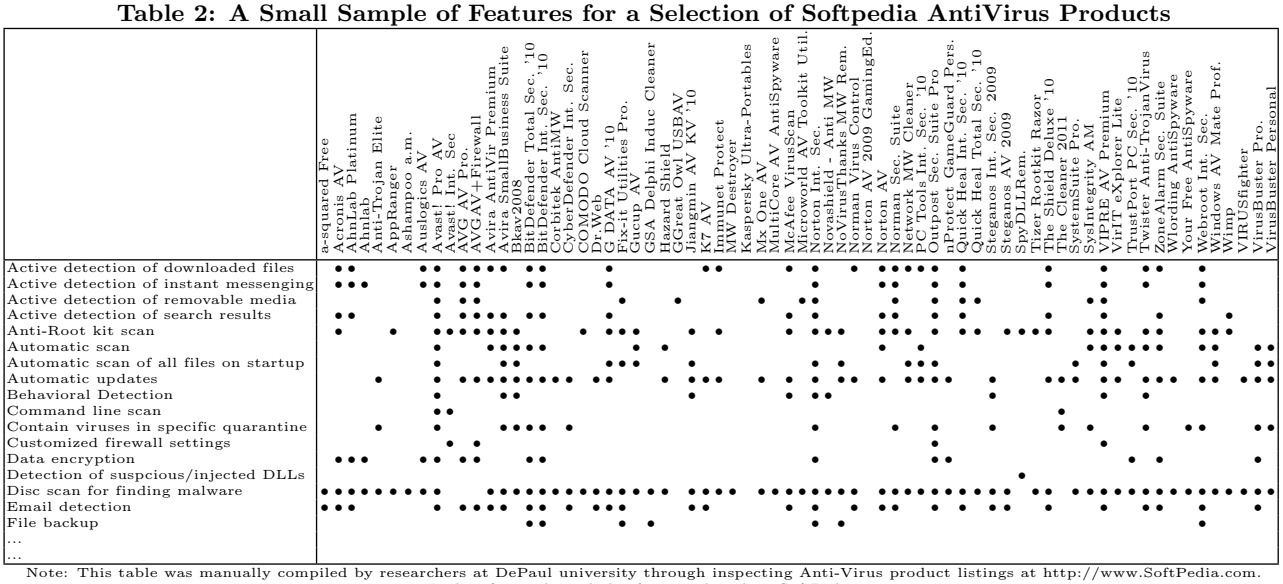
Recommendations are made in two different ways. First, the feature recommender utilizes a *Content Based recommender* based on the standard kNN algorithm to generate feature recommendations. Secondly, it utilizes association rules which are often able to identify hidden affinities between features across products. Recommendations based on association rules can differ from and thus be complementary to the recommendations made by the kNN approach. Furthermore, association rules can sometimes be used to generate recommendations from relatively sparse product profiles, which can be useful if a user provides only limited information for an initial product description. Figure 2 illustrates a feature recommendation scenario for an anti-virus product. An initial product description is mapped to four features in the recommender's knowledge base related to *spam detection, disk scans, virus definitions*, and *virus databases*, which serve as seeds for generating feature recommendations. User feedback is used to drive and refine future recommendations.

The remainder of this paper is structured as follows.

Table 1: Product Categories

Product Category	Product Count	Feature Count
Antivirus	165	40
Audio-Players	283	46
Bookmark-Managers	92	33
Browsers	205	41
Coding-languages-Compilers	154	50
Compression-tools	159	40
Debuggers-Decompilers-Dissasemblers	118	46
Digital-Photo-Tools	314	46
Download-Managers	207	41
FTP-Servers	59	38
File-Sharing	278	44
File-managers	260	46
Multimedia-IPOD-tools	97	34
Office-suites	50	52
Password-Managers-Generators	281	38
Popup-Ad-Spyware-Blockers	211	35
Search-engine-tools-submitting	198	41
Text-editors	333	48
Video-Recording	150	37
WebCam	129	40

Table 2: A Small Sample of Features for a Selection of Softpedia AntiVirus Products



Note: This table was manually compiled by researchers at DePaul university through inspecting Anti-Virus product listings at http://www.SoftPedia.com. It therefore only includes features listed on SoftPedia.

Sections 3 and 4 describe the process for mining and discovering features using the Incremental Diffusive Clustering method. Sections 5 and 6 describe kNN and association rule recommendation algorithms. Section 7 describes empirical experiments that were conducted to evaluate the recommender system, while Section 8 provides an anecdotal example. Section 9 describes related work, and Section 10 summarizes the contributions of this paper and discusses future work.

3. MINING RAW PRODUCT DATA

All of the feature descriptors used in this paper were mined from *Softpedia.com*. However, our approach is easily adapted for use with product descriptions from other online sources. Softpedia provides descriptions for a wide variety of software products including Windows, Linux, Mac, and mobile applications. As of August, 2010, it included over 796,536 different software applications, with over 1.5 billion downloads, categorized under 9 primary groupings, 292 sub categories, and 1,096 product types. Most products include a section of *feature descriptors* formatted in a bullet-point list format. As formatting is fairly consistent, we utilized the Screen-Scraper© utility, to automate the extraction of the raw product features.

As Softpedia, and other similar repositories, describe products for marketing purposes and do not provide full technical specifications, the feature lists for each individual product cannot be expected to be complete. This is an important observation that impacts the feature mining process. Whereas, previous approaches have used a small number of relatively complete specifications [8, 3, 16, 25, 33], our approach, which analyzes thousands of individually incomplete specifications, is able to automatically infer associations due to the sheer quantity of the analyzed data. For experimental purposes, descriptors were mined from the 20 Softpedia product categories shown in Table 1.

Table 2 illustrates a small subset of the features manually identified for the Softpedia Anti-virus software products. It shows that most of the analyzed Anti-virus systems include *core* features such as *Automatic Updates* or *malware scan*, as well as variants such as *file backup* and *data encryption* which are found in only a small subset of products. Furthermore, certain features such as *Anti-root kit scan* are found primarily in anti-virus software applications, while others, such as *Password Manager* are common across many different product categories. Various associations can be observed between features. For example, 80% of products containing *parental controls* also track *web history*, while conversely 77% of Anti-Virus products that provide *integration with 3rd party software* do NOT support *protected files*. Understanding such association and disassociation rules can help improve the organization of feature recommendations.

4. FEATURE DISCOVERY

Our approach utilizes an incremental diffusive clustering (IDC) algorithm that we had previously developed to detect recurring problems in flight anomaly reports [14, 9]. In this case it was used to construct a domain model of common and variant features. Following a preprocessing phase, the algorithm incrementally identifies features by repeatedly executing the following steps: (i) clustering the descriptors into candidate features, (ii) selecting and retaining the "best" feature, (iii) identifying dominant terms and removing them from all descriptors to produce new "reduced" descriptors, and then (iv) repeating steps i-iii using the increasingly reduced descriptors until the targeted number of features have been generated. The concept behind the clustering algorithm is to progressively remove terms representing dominant topics around which features have already formed, thereby allowing latent topics to emerge.

4.1 Incremental Diffusive Clustering

Feature descriptors are first preprocessed by removing commonly occurring words and then by stemming the remaining words to their root form (i.e., terms). Given the dictionary of all such terms $T = \{t_1, t_2, \ldots, t_W\}$, each raw feature, f_i is represented as a vector of terms, $v_i =$

$(f_{i,1}, f_{i,2}, \ldots, f_{i,W})$ where $f_{i,j}$ is a term weight representing the number of occurrences of term t_j in the raw feature description f_i. These term weights are then transformed using a standard term-frequency, inverse document frequency (tf-idf) approach such that, $tf\text{-}idf(f_{i,j}) = f_{i,j} \cdot log_2(D/df_j)$ where D represents the total number of raw feature descriptions, and df_j represents the number of raw feature descriptions containing term t_j. Finally, the transformed vector (with tf-idf weights) is normalized to a unit vector resulting in the vector $V_i = (F_{i,1}, F_{i,2}, \ldots, F_{i,W})$. The following steps are then executed to identify features.

Step 1: Granularity determination

To determine how many features to generate for a given product category, we adopted a modified version of Can's metric [5] which considers the degree to which each raw feature description differentiates itself from other raw features. The ideal number of clusters K is computed as follows:

$$K = \sum_{v_i \in F} \sum_{j=1}^{W} \frac{f_{i,j}}{|v_i|} \cdot \frac{f_{i,j}}{N_j} = \sum_{v_i \in F} \frac{1}{|v_i|} \sum_{j=1}^{W} \frac{f_{i,j}^2}{N_j} \qquad (1)$$

where N_j represents the total number of occurrences of term t_j. Through observing the results of preliminary clustering experiments, we modified the algorithm to exclude terms for which N_j fell below a threshold determined empirically to be $0.0075D$. Introducing this threshold improved cluster quality by removing low frequency terms that had little effect in representing potential features.

Step 2: Clustering feature descriptors

Our Incremental Diffusive Clustering (IDC) method uses a consensus clustering approach in which a predefined number of candidate clusterings (in this case 25) are generated and then integrated into a final clustering through use of a voting scheme [12]. Although not reported in this paper, we previously conducted an extensive comparison of different approaches for clustering requirements and features [11]. Techniques included Latent Semantic Analysis (LSA), Latent Dirichlet allocation (LDA), agglomerative hierarchical methods, and different variants of K-Means clustering methods. Our experiments showed that consensus clustering methods, used in conjunction with SPK outperformed other methods on the tested datasets. Based on these findings [12] we therefore adopted a consensus based approach for the feature recommender system, in which each candidate clustering is generated by selecting 75% of the original descriptors, clustering them using the standard SPK algorithm [10], and then classifying the remaining 25% into their most similar clusters. Multiple candidate clusterings are then reconciled to generate a final set of clusters. Although the consensus clustering algorithm has a relatively long running time, it produces results that are consistently of higher quality than the average SPK clustering, and invariably close to the optimal quality achievable by an individual SPK clustering [12].

Step 3: Selecting the best cluster

In each iteration IDC selects and promotes the "best" cluster to the status of a feature. Based on initial observations, the best cluster from a set $L = \{C_1, C_2, \ldots, C_K\}$, is one that has high levels of cohesion with broad coverage of the topic. To measure cohesion for a cluster $C_i = \{V_{i,1}, V_{i,2}, \ldots, V_{i,r}\}$ with centroid $\overline{C_i}$, the similarity between the features and their associated centroids is computed and averaged as $(\sum_{j=1}^{r} sim(V_{i,j}, \overline{C_i}))/r$. while topic coverage is computed as $\sum_{j=1}^{r} sim(V_{i,j}, \overline{C_i})$. Cohesion and topic coverage scores are added together, and the cluster with the highest combined score is selected as the "best" cluster and promoted to the status of a feature.

Step 4: Removing dominant terms

To remove dominant terms, all terms exhibiting weights above a predefined threshold (0.15) in the vector representing the centroid of the "best" cluster are selected. Since the centroids are also normalized, this threshold is held constant. These terms are then removed from all descriptors in the dataset. For example, if dominant terms are identified as *instant*, *messag* and *encrypt* (shown in stemmed form), then a descriptor originally specified as *Encrypt email messages before transmission* is reduced to *messages before transmission* (with the word *before* also removed as a stop word). Future rounds of clustering are then performed on the reduced version of descriptors.

Steps 2-4 are repeated until the targeted number of features, determined previously in Step 1, have been identified.

Step 5: Post processing

A post-processing step is executed to optimize the identified features. This step involves the four tasks of (i) removing misfits, (ii) recomputing centroids, (iii) checking for missing members, and finally (iv) merging similar features. Misfits are removed by computing the similarity score between each centroid and its associated descriptors, and then removing any feature descriptions which fall below a given threshold value (set to 0.35 based on empirical observations). Centroids are then repositioned according to the remaining descriptors using the same SPK algorithm adopted in the initial clustering step. Missing descriptors are identified through recomputing the cosine similarity between every raw feature descriptor not currently assigned to a feature, and the centroid of that feature. Any feature exhibiting a similarity score higher than a given threshold (set to 0.35) is added to that feature. Finally, similar features are merged by computing the cosine similarity between each pair of centroids. Any pair of features exhibiting a score above a given threshold (set to 0.55) is merged into a single feature.

Step 6: Feature naming

Each feature is named through identifying the *medoid*, defined as the descriptor that is most representative of the feature's theme. The medoid is identified by first computing the cosine similarity between each descriptor and the centroid of the cluster, and then summing up the different weighted values in the descriptor's term vector for all values above a certain threshold (0.1). Both scores are normalized and then added together for each descriptor. The descriptor scoring the highest value is selected as the feature name. This approach produces quite meaningful names. As an example, a feature based on the theme of *updat, databas, automat, viru* was subsequently named *Virus definition update and automatic update supported.*

4.2 Merging Category Clusters

Our approach involves processing each product category separately and then merging them into a single product-by-feature matrix. This directly mitigates scalability issues of dealing with large numbers of features, and has the additional extensibility benefit of allowing new product categories to be added incrementally. Merging is accomplished through computing the cosine similarity between each pair of features, and then merging features exhibiting scores of 0.6 or higher. The end result is illustrated by the three features

Table 3: Feature analysis

Feature	Easy to use and friendly user Interface	Rotating Cropping and Resizing images	Save compressed files
# of categories containing that feature	12	1	2
Most common Category (MCC)	File Sharing	Photo Tools	Video Recording
% of products with that feature in MCC	0.569	0.444	0.277

depicted in Table 3. The first feature *Easy to use and friendly user interface* is an example of a merged feature that cuts across 12 of the 20 analyzed product categories. In contrast, the other two features are specific to one or two product categories only.

4.3 Evaluating Feature Quality

The quality and completeness of features generated for Anti-Virus software, multi-media iPod applications, and Photography applications, were independently evaluated by three graduate level researchers at DePaul, and compared against feature models that had been manually developed as "answer sets" by independent analysts. Each assessor judged whether each automatically generated feature represented (i) a clearly defined feature that matched a feature in the answer set, (ii) A meaningful feature that was not included in the answer set, (iii) a non-unique feature that overlapped with a previously generated feature, or (iv) an ill-formed feature that was non-cohesive or otherwise did not make sense. They were also asked to rate the feature name as (i)Excellent, (ii) Good, (iii) Poor, or (iv) Completely inadequate; where an excellent feature was described as having a realistic and professional sounding feature name, and an inadequate feature name exhibited significant problems such as being too lengthy or grammatically significantly incomplete. Finally, each evaluator was asked to either match each feature in the manually created answer set with a similar feature generated by IDC, or to mark it as missing.

Results from the evaluation are shown in Table 4 and show that while our approach discovered only about 50% of the features in the answer set, it also discovered an equivalent number of unique features that were missed in the manually constructed answer set. This occurred despite the fact that analysts spent from 10-18 hours constructing each answer set. Our initial observations suggest that additional features could have been detected by expanding the scope of the screen scraper to mine broader product descriptions beyond items listed as features. It is also worth noting that our naming algorithm performed very well, delivering almost 90% of features there were categorized as Good or Excellent. Only 2% of names were deemed inadequate.

5. FEATURE RECOMMENDATIONS USING STANDARD KNN

Once features have been identified, a product-by-feature matrix is generated, $M := (m_{i,j})_{P \times F}$, where P represents the number of products and F the number of identified features. In a typical domain analysis, features are modeled using a specific notation that depicts commonalities, variabilities, mandatory, and optional features. In our approach, this knowledge is represented in the product-by-feature matrix and the frequent item set graph. From these data structures it is possible to infer some of the information that is typically captured as part of a more traditional feature diagram [17], namely commonalities, variants, and cross-cutting features. For this reason, our approach does not explicitly construct a feature diagram.

Table 4: User Evaluation of Feature Quality and Completeness

	Virus	iPod	Photo	Overall
Total features in answer set	36	30	38	104
In both auto gen. and answer set	15	15	20	50
Answer set only	21	15	18	54
Auto generated only	15	15	19	49
Redundant features	0	3	1	4
Ill-formed features	3	2	2	7
Excellent name	0.71	0.50	0.69	0.63
Good name	0.15	0.36	0.27	0.26
Poor name	0.12	0.11	0.04	0.09
Inadequate name	0.03	0.03	0.00	0.02

Several steps are needed to generate a feature recommendation. These include seeding the recommendation request, creating an initial product profile, using association rule mining to make initial recommendations and then to augment the product profile, and finally recommending additional features using a Content Based recommendation algorithm based on standard kNN.

5.1 Creating a new product profile

An initial description or list of features is provided for the new product, which is then automatically parsed to break the text into segments (sentences or phrases), remove unimportant terms (i.e., stopwords), and stem words to root forms. The cosine similarity metric is used to match individual segments to previously mined features. If insufficient matches are found no recommendations are generated, and the user is notified that either the targeted product is outside the knowledge of the recommender system, or that the initial product description contains insufficient detail to launch a recommendation. For purposes of this paper, we established a threshold score of 0.6 for the cosine similarity, and then required at least 5 product-specific features to be matched to known features. The new product profile is appended as a row in the product-by-feature matrix.

In many cases, the initial product profiles may be rather sparse and unable to provide sufficient information to generate high quality recommendations using more traditional techniques such as content or collaborative recommenders. The association-based recommender system described in section 6 can be used to identify additional features from high confidence association rules. These recommended features can also allow a user to explore the possible feature space for the product. The user can be asked to provide feedback on these recommendations, and then accepted features can be used to augment the product profile in preparation for additional feature recommendations.

5.2 Content-based recommendation

We adopted a standard K-Nearest Neighbor (kNN) learning strategy to design a content-based recommender algorithm. This approach has been shown to perform well in

our previous work in forum recommendations [6, 7]. In this context, the kNN algorithm computes a feature-based similarity measure between a new product and each existing product. The top k most similar products are considered *neighbors* of the new product. It then uses information from these neighbors to infer other potentially interesting features and to make recommendations. In the first step, the product similarity $productSim(p, n)$ between a new product p and each existing product n is computed using the binary equivalent of the cosine similarity as follows:

$$productSim(p, n) = \frac{|F_p \cap F_n|}{\sqrt{|F_p| \cdot |F_n|}} \quad (2)$$

where F_p denotes the set of features of product p [32]. This metric generates numbers between 0 and 1, where products with identical features score 1, and products with no common features score 0. $productSim(p, n)$ is computed between the new product and all previously mined products. A neighborhood is then computed for the new product p by selecting the top k most similar products. Based on an initial series of experiments, we set the value of k to 10. The second key computation of the algorithm predicts the likelihood that stakeholders for product p will be interested in including a new feature f. For this, a variation of the standard prediction formula [31] is used:

$$pred(p, f) = \frac{\sum_{n \in nbr(p)} productSim(p, n) \cdot m_{n,f}}{\sum_{n \in nbr(p)} productSim(p, n)} \quad (3)$$

where $n \in nbr(p)$ depicts that n is a neighbor of p, and $m_{n,f}$ is an entry in the matrix M indicating whether product n contains feature f. In general, prediction scores will be computed for each candidate feature, and the features with highest predictions will be recommended.

6. FEATURE RECOMMENDATIONS USING ASSOCIATION RULES

Features within product categories exhibit various kinds of associations or disassociations which can be leveraged to improve the accuracy of feature recommendations. Association rule discovery techniques, such as the Apriori algorithm [2], were initially developed to discover groups of products that buyers typically purchase together. Association rules identify groups of items based on patterns of co-occurrence across transactions. Our approach applies association rule mining to discover latent relationships between features within products. In this context each product is viewed as a "transaction", and association rules are generated among sets of features that commonly occur together among a significant number of products.

Given a set of product profiles P and a set of features $F = \{F_1, F_2, \cdots, F_k\}$, and a feature set $fs \subseteq F$, let $P_{fs} \subseteq P$ be the set of products that have all the features in fs. The *support* of the feature set fs is defined as $\sigma(fs) =\mid P_{fs} \mid /|P|$. Feature sets that satisfy a predefined support threshold are referred to as *frequent item sets* (in the following we will refer to these as *frequent feature sets*).

An association rule r is expressed in the form $X \Longrightarrow Y(\sigma_r, \alpha_r)$, where $X \subseteq P$ and $Y \subseteq P$ are feature sets, σ_r is the support of the feature set $X \cup Y$, and α_r is the *confidence* for the rule r given by $\sigma(X \cup Y)/\sigma(X))$. The discovery of association rules from a transaction database involves two main parts: the discovery of frequent feature sets

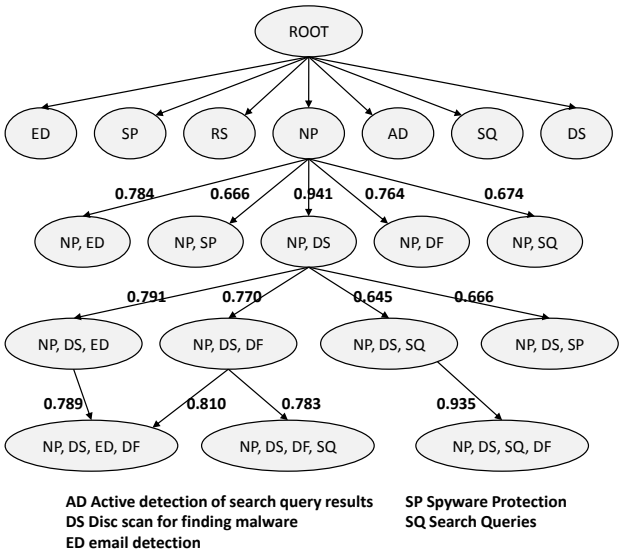


Figure 3: A Subset of a Frequent Itemset Graph for the *Network Intrusion Protection* augmented with confidence scores

(i.e., itemsets which satisfy a minimum support threshold) and the discovery of association rules from these frequent feature sets which satisfy a minimum confidence threshold. Note that association rules are only generated from frequent features sets, so features that do not appear in at least one frequent itemsets are filtered out and will not appear in any rules. The strength of a discovered rule is measured based on the support of the underlying feature set as well as the confidence of the rule.

Among other types of analysis, association rules have been used as the basis for a class of recommender systems [20, 23, 30], especially in e-commerce and intelligent Web applications. The basic approach is to match a partial profile comprised of a set of items against the antecedent X of each discovered rule, and then sort the items on the right hand side of the matching rules according to the confidence values. The top ranked items from this list form the recommendation set. In our case, we start with an initial product profile (a set of features) and use the association-based algorithm to recommend new features. The new features, in turn, can be used either as stand-alone recommendations, or as a way to enrich the initial product profile. It should be noted that. if there is not enough support for a particular feature, that feature will never appear in any frequent feature set, and will therefore never be recommended based on association rules. In our approach, we retain a relatively high support threshold value in order to guarantee that only relatively strong associations are used in the initial recommendations.

To facilitate the search for matching association rules, the frequent feature sets are stored in a directed acyclic graph, called a Frequent Itemset Graph (FIG)[23, 29]. The graph is organized into levels from 0 to k, where k is the maximum size among all discovered frequent feature sets. Each node at depth d in the graph corresponds to a feature set I of size d and is linked to features sets of size $d + 1$ that contain I at the next level. The root node at level 0

corresponds to the empty features set. Each node also stores the support value of the corresponding frequent feature set. The FIG represents the aggregate model learned in the first phase of association rule discovery performed offline. The recommendations are, however, generated at query time by performing depth-first searches of the FIG. Given an initial product profile comprised of a feature set f, the algorithm performs a depth-first search of the graph to level $|f|$. Each candidate recommendation r is a feature contained in a frequent feature set $f \cup \{r\}$ at level $|f+1|$. For each such child node of f, the feature r is added to the recommendation set if the support ratio $\sigma(f \cup \{r\})/\sigma\{f\}$, which is the confidence of the association rule $f \Rightarrow \{r\}$, is greater than or equal to a pre-specified minimum confidence threshold. This process is repeated for each subset of the initial feature set f and conflicting candidate recommendations are resolved by retaining the highest confidence values.

Figure 3 shows a small subset of frequent feature sets mined from the anti-virus software features. The displayed itemset graph shows features associated with *network intrusion detection*. For example, one rule emanating from this graph states that if *network intrusion detection (NP)* and *disk scan for finding malware (DS)* features are found in a product, then we have a confidence of 0.791 that the product will also contain an *email detection* feature.

7. EVALUATION

Evaluating a recommender system is a non-trivial task. The ideal approach would involve conducting a study of several software projects to observe whether any of the recommended features were ultimately included in the final product. However this type of evaluation would need to be conducted over a long period of time and would require multiple software projects.

Fortunately there are several commonly used statistical methods for evaluating recommender systems [15]. One such method is a leave-one-out cross validation approach, which if applied to the feature recommendation problem would involve systematically removing a random feature from each product profile, executing the recommender system to generate a recommendation feature set of size N, ranked according to prediction values, and determining if the removed feature is part of the recommendation set.

Results for leave-one-out cross validation recommendation experiments are typically evaluated by computing the *Hit Ratio*, which computes the probability that a given feature is recommended as part of the top N recommendations produced by the system. Specifically, for each product p in the test set, a feature f_p is randomly removed from the product profile and a recommendation set $R_N(p)$, comprised of the top ranked N recommended features, is produced using the remaining features of p. If $f_p \in R_N(p)$, that is considered a *hit* for that product. Suppose P is the set of products used for evaluation. The hit-ratio of the recommendation algorithm relative to a recommendation set size of N, is computed as: $hr(N) = |p \in P : f_p \in R_N(p)|/|P|$. Typically, hit ratio values are plotted against different values of N. A hit ratio value of 1.0 indicates that the algorithm was able to always recommend the hidden feature, whereas a hit ratio of 0.0 indicates that the algorithm was not able to recommend any of the hidden features. Note that hit ratio increases as the recommendation set size (N) increases. In particular, if N is the total number of possible features, the hit ratio will be 1. Therefore, ideally a recommender system should be able to achieve relatively high hit ratios even when the recommendation set is small.

7.1 Experimental Design

A series of three different experiments were designed to evaluate a realistic use of the feature recommender system, in which a domain analyst might provide a partial description of a product, and then request recommendations for additional features. These three experiments evaluate (i) the use of kNN with various sized product profiles, (ii) the use of association rule mining with incomplete product profiles, and finally (iii) a hybrid approach that utilizes association rule mining to augment incomplete profiles and then uses kNN to make additional recommendations. All experiments described in this paper share a common experimental design so that results can be compared across experiments.

1. Experiments were set up as five-fold, cross-validations, in which 20% of products from each category were assigned to each set. In each of five runs, one of the sets served as the testing set, while the remaining four were combined into a training set. After five runs, each set had served as the test set one time. It should be noted that this five-fold approach was only necessary for the two experiments involving association rule mining; however it was applied across all of the experiments so that results could be more easily compared.

2. For each product in the test set, L features were randomly selected and retained as the starting product profile. One of the remaining features was then randomly selected as a *target* item and removed from the initial profile. In each case, the recommender system was evaluated with respect to whether it was able to recommend back this targeted item. All other known features were removed from the product profile and not used in the current experiment.

3. To support experiments that included 3, 4, and 5 features in the initial profile, plus one target item, only products with 6 or more features were included in the experiment. This resulted in 1,364 product profiles in the product-by-feature matrix.

7.1.1 Experiment 1: Varying profile size

For the first experiment the previously described experimental design was adopted in order to evaluate the use of the kNN algorithm. The experiment was repeated three times with profile sizes of $L = 3$, 4, and 5. Results are reported in the hit ratio graph shown in Figure 4. As expected the recommender performed better given larger initial profiles. It is worth noting that for all three experiments 50% of features were presented in the top 17 features. For the largest training set (L5F1) over 80% features were returned within the top 30 recommendations.

7.1.2 Experiment 2: Association Rules

The second experiment evaluated the use of association rules for making feature recommendations. Again, the previously defined five-fold experiment was adopted. In each of the five runs of the experiment, a frequent item set graph was constructed from the product profiles in the training set.

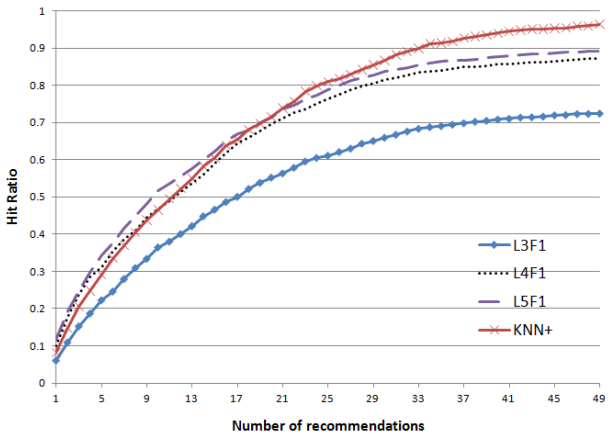


Figure 4: Hit ratio for kNN Recommendations with initial profile of size 3, 4, 5 and with profiles augmented by association rules (kNN+)

Initial profile sizes were set at $L = 3$ for all products in the test set. Association rules were then generated with respect to these initial profile, and used to create recommendations at confidence values of 0.6 or higher. As recommendations based on association rules tend to be fairly precise but also incomplete, the results of this experiment were evaluated by measuring precision and coverage of the recommendations, where precision is computed as the fraction of recommended features that were marked as *target* features, and coverage is computed as the fraction of product profiles which received recommendations. We also measured recall, which as expected was relatively low. Results are depicted in Figure 5 and show that recommendation precision increases as the confidence threshold is raised until a point of diminishing return is reached. At lower levels of precision, approximately 45% of product profiles received recommendations, however this number dropped to only 1% as confidence scores and precision values increased. These results suggest that although association rule mining can be used to generate a small, yet relatively precise set of feature recommendations, it would not be effective on its own for finding a more complete set of features.

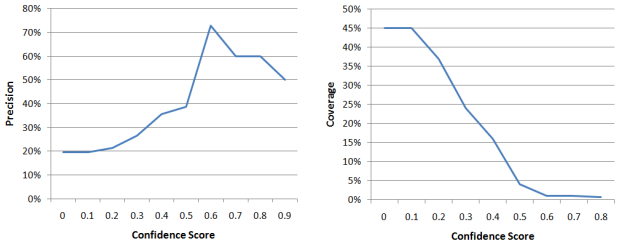


Figure 5: Precision and Coverage achieved using Association Rules

7.1.3 *Experiment 3: Augmented Profile*

One of the problems frequently experienced with typical recommender systems, is the *cold start* problem that occurs when recommendations must be made for a user for whom no profile exists. In the case of the feature recommender, although an initial profile can be extracted from a visionary description of the product, the profile may still be relatively terse, resulting in a sparse profile. To address this problem we investigated a novel hybrid approach in which Association Rules are first used to augment an initially sparse matrix, and then the standard kNN algorithm is applied to the augmented profile. The following experiment was conducted to evaluate whether this approach might lead to improved feature recommendations.

The first part of the experiment followed all of the steps described in experiment 2. However once the association rules had been used to generate recommendations, these recommendations were all automatically accepted and used to augment the initial product profile of size $L = 3$. The kNN leave-one-out experiment described in Experiment 2 was then conducted against the augmented profiles.

Results are also reported in Figure 4 as $KNN+$. These results should be compared to $L3F1$, as both cases had initial profile sizes of L=3. Using the association rules to augment the originally sparse product profile led to significant improvements. For example $KNN+$ showed improvements of approximately 10% in the top 5 recommendations, 16% in the top ten, and greater than 25% in the top 50 recommendations.

7.2 Analysis of Results

Results from these experiments have shown that the kNN recommender system has the ability to recommend useful features to users. It performs better as the size of the product profile increases. Furthermore, when the product profile is sparse, association rules can be used quite effectively to enrich the profile and improve the performance of the kNN recommender.

There are a number of threats to validity for our study. First, all of the feature descriptors are mined from a single repository. However this repository is very large, containing many hundreds of thousands of product descriptions. We anticipate that our approach will generalize to other sources of product descriptions simply by customizing the screen scraping and feature extraction process. A threat to the generalization of our approach is that the software applications on Softpedia represent small to medium sized applications, each of which tends to perform a very specific task. We therefore cannot yet claim that our approach would apply for large software systems, in which numerous modules must interact in carefully defined ways. Nevertheless, the feature recommender system could still prove useful for proposing features for various components of such larger software systems. Our approach is also limited by the scope of the products described in online repositories. However we leave it to future work to explore broadening the descriptor mining process to discover and incorporate a much larger set of product features.

8. AN ANECDOTAL EXAMPLE

To illustrate the use of the feature recommender in a more anecdotal way, Table 5 provides a simple example of recommendations made as a result of seeding a product profile with four initial features. Three of these features were related to managing music, while one was a general usability feature. The association rules identified one additional fea-

ture related to dragging and dropping of files into the play list with a high level of confidence. This feature was added to the product profile. The kNN recommender then proposed several additional recommendations, most of which appear to be very viable suggestions. The one exception is *Create Photo Albums* which appears unrelated, but introduces the serendipitous idea of creating music albums. In general, this example demonstrates that the feature recommender took a relatively sparse initial product profile and made a series of viable and useful feature recommendations.

Table 5: Anecdotal Example

Initial Profile	-Easy to use and friendly User Interface -Listen to Internet radio streams -Supported audio file formats MP3 -WMA OGG FLAC MP4 M4A
Features recommended by Association rules	Drag and drop files into the playlist
Features recommended by KNN	-Customize the program with user interface skins -Audio formats supported WAV OGG MP1 MP2 MP3 MOD XM IT S3M MTM UMX MO3 -10 band equalizer with presets -Play Media & Video Files mp3 wav wma mpg mpeg avi -Create Photo Albums -Export albums to burn to CD DVD -Support Winamp2 x and Sonique Visualizations and Plugins -Easy To Use Only 2 Click

9. RELATED WORK

This work bridges the gap between automated feature detection and recommender systems. We therefore provide a brief background survey on each of these areas.

Several researchers have previously investigated the use of information retrieval methods for constructing feature models. For example, the Domain Analysis and Reuse Environment (DARE) [13] uses semi-automated tools to extract domain vocabulary from text sources, and then identifies common domain entities, functions, and objects, by clustering around related words and phrases. Chen et al. [8] manually construct requirements relationship graphs (RRG) from several different requirements specifications and then use clustering techniques to merge them into a single domain tree. Vander Alves et.al. [3] utilized the Vector Space Model (VSM) and Latent Semantic Analysis (LSA) to determine the similarity between requirements and generate an association matrix which is then clustered. A merging step is then executed to create the entire domain feature model. Noppen et al. [16] extended this work by including fuzzy sets in the framework to allow individual requirements to be associated into multiple features. Niu and Easterbrook [24, 25] developed an on-demand clustering framework that provided semi-automatic support for analyzing functional requirements in a product line [22]. Weston et al. [33] introduced a tool named ArborCraft, which creates feature models from requirements specifications by utilizing clustering methods to identify initial features and then a vocabulary lexicon and grammatical pattern matching to identify feature variants. The primary limitations of these approaches are their reliance upon existing requirements specifications, and the constraints associated with mining features from only a small handful of specifications. On the other hand, such requirements specifications provide a deeper perspective of the product than our approach has to infer through analyzing hundreds of different product specifications.

The field of recommender systems has also been studied extensively, but mostly within the context of e-commerce systems, where numerous algorithms have been developed to model user preferences and create predictions. These algorithms vary greatly, depending on the type of data they use as an input to create the recommendations. For example, some use content information about the items [26], or collaborative data of other users' ratings [31], or knowledge rules of the domain [27], or hybrid approaches [28]. Substantial amounts of work have been done in the area of evaluating recommender systems [15] and on newer highly efficient algorithms such as those based on matrix factorization [19]. Despite the proliferation of both of these fields, there has been very little work combining recommender systems and requirements engineering. Work in this area has focused on recommending topics of interest in large-scale online requirements forums [7], and a high-level overview of possible usages and applications of recommender systems in this domain [21]. The use of recommender systems in this paper builds on the substantial background of research in this area.

10. CONCLUSIONS

This paper has presented a novel approach for mining and recommending product specific features, and therefore providing support for the critical software engineering task of domain analysis. It differs significantly from prior work in the area of feature mining, which relies upon the availability of detailed requirements specifications or documentation. In contrast, our approach discovers features and their associations through mining publicly available repositories of product descriptions. To achieve this we have introduced a novel Incremental Diffusive Clustering algorithm, which is well suited to discovering features. Another novelty of this work is the hybrid approach which uses association rule mining to augment an initial profile, and then uses the standard kNN approach to make additional recommendations. Finally, to the best of our knowledge, prior work in the area of feature mining has focused on automating the discovery of features from requirements specifications, but has not developed a corresponding recommending system. In our case, the statistical engine behind the recommender system is enabled because of the large amount of supporting product data.

The findings reported in this paper introduce numerous additional questions and areas of future work. In the area of feature mining, we plan to extend the work to mine feature descriptions from general product descriptions instead of limiting them to the bulleted lists found in SoftPedia. We believe this will enrich the profiles of each individual product. Furthermore, future work will also include using both positive as well as negative associations among features to enhance kNN recommendations.

11. ACKNOWLEDGMENTS

The work described in this paper was partially funded by grant III:0916852 from the National Science Foundation.

12. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *1993 ACM SIGMOD Conf. on Management of Data*, pages 207–216, 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Intn'l Conf. on Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, September 1994.

[3] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, K. Pohl, and A. Rummler. An exploratory study of information retrieval techniques in domain analysis. *Proc. of the 12th Intn'l Software Product Line Conf.*, pages 67–76, 2008.

[4] G. Arango and R. Prieto-Diaz. *Domain Analysis: Acquisition of Reusable Information for Software Construction*. IEEE Comp. Society Press, May 1989.

[5] F. Can and E. Ozkarahan. Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases. *ACM Trans. Database Syst.*, pages 483–517, 1990.

[6] C. Castro-Herrera, J. Cleland-Huang, and B. Mobasher. Enhancing stakeholder profiles to improve recommendations in online requirements elicitation. In *Requirements Eng., IEEE Intn'l Conf. on*, pages 37–46, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[7] C. Castro-Herrera, C. Duan, J. Cleland-Huang, and B. Mobasher. A recommender system for requirements elicitation in large-scale software projects. *Proc. of the 2009 ACM Symp. on Applied Computing*, pages 1419–1426, 2009.

[8] K. Chen, W. Zhang, H. Zhao, and H. Mei. An approach to constructing feature models based on requirements clustering. *Requirements Engineering, IEEE International Conference on*, 0:31–40, 2005.

[9] J. Cleland-Huang and B. Mobasher. Automated detection of recurring faults in problem anomoly reports. *SERC Report to Lockheed Martin*, 2009.

[10] I. Dhillon and D. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42:143–175, 2001.

[11] C. Duan. Clustering and its application in requirements eng. *Technical Report, School of Computing, DePaul University*, 2008.

[12] C. Duan, J. Cleland-Huang, and B. Mobasher. A consensus based approach to constrained clustering of software requirements. *ACM Conf. on Information and Knowledge Management*, pages 1073–1082, 2008.

[13] W. Frakes, R. Prieto-Diaz, and C. Fox. Dare: Domain analysis and reuse environment. *Annals of Software Eng.*, 5, 1998.

[14] H.Dumitru, A.Czauderna, and J.Cleland-Huang. Automated mining of cross-cutting concerns from problem reports and requirements specifications. *Argonne Undergraduate Symp.*, 2009.

[15] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. on Info. Sys.*, 22:5–23, 2004.

[16] J.Noppen, P. van den Broek, N. Weston, and A. Rashid. Modelling imperfect product line requirements with fuzzy feature diagrams. *VaMoS*, pages 93–102, 2009.

[17] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented domain analysis (foda) feasibility study. 1990.

[18] I. Karlsen, N. Maiden, and A. Kerne. Inventing requirements with creativity support tools. In *Working Conf. on Requirements Eng. for Software Quality*, pages 162–174, June 2009.

[19] J. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, 2009.

[20] W. Lin, S. A. Alvarez, and C. Ruiz. Efficient adaptive-support association rule mining for recommender systems. *Data Mining and Knowledge Discovery*, 6:83–105, 2002.

[21] W. Maalej and A. Thurimella. Towards a research agenda for recommendation systems in requirements eng.,. *Proc. of the 2nd Intn'l Workshop on Managing Requirements Knowledge (MARK)*, pages 32–39, 2009.

[22] M.Moon, K.Yeom, and H.S.Chae. An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. *IEEE Trans. on Software Eng.*, 31:551–569, 2005.

[23] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa. Effective personalization based on association rule discovery from web usage data. In *3rd ACM Workshop on Web Information and Data Management (WIDM01)*, Atlanta, November 2001.

[24] N.Niu and S.Easterbrook. Extracting and modeling product line functional requirements. *Proc. of the 16th Intn'l Requirements Eng. Conf.*, 2008.

[25] N.Niu and S.Easterbrook. On-demand cluster analysis for product line functional requirements. *Proc. of the 12th Intn'l Software Product Line Conf.*, pages 87–96, 2008.

[26] M. Pazzani and D. Billsus. Content-based recommendation systems. *The Adaptive Web*, pages 325–341, 2007.

[27] R.Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Info. Sys.*, 69, 2000.

[28] R.Burke. Hybrid rec. systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12:331–370, 2002.

[29] J. Sandvig, B. Mobasher, and R. Burke. Robustness of collaborative recommendation based on association rule mining. *Proc. of Recommender Systems*, 2007.

[30] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommender algorithms for e-commerce. In *Proc. of the 2nd ACM E-Commerce Conf. (EC'00)*, pages 158–167, Minneapolis, MN, October 2000.

[31] J. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. *The Adaptive Web*, page 291, 2007.

[32] E. Spertus, M. Sahami, and O. Buyukkokten. Evaluating similarity measures: a large-scale study in the orkut social network. pages 678–684, Chicago, Illinois, USA, 2005. ACM.

[33] N. Weston, R. Chitchyan, and A. Rashid. A framework for constructing semantically composable feature models from natural language requirements. *Proc. of the 13th Intn'l Software Product Line Conf.*, 2009.