Supporting Domain Analysis through Mining and Recommending Features from Online Product Listings

Negar Hariri, Carlos Castro-Herrera, Mehdi Mirakhorli, *Student Member, IEEE*, Jane Cleland-Huang, *Member, IEEE*, Bamshad Mobasher, *Member, IEEE*

Abstract—Domain analysis is a labor intensive task in which related software systems are analyzed to discover their common and variable parts. Many software projects include extensive domain analysis activities, intended to jumpstart the requirements process through identifying potential features. In this paper, we present a recommender system that is designed to reduce the human effort of performing domain analysis. Our approach relies on data mining techniques to discover common features across products as well as relationships among those features. We use a novel incremental diffusive algorithm to extract features from online product descriptions, and then employ association rule mining and the k-Nearest-Neighbor machine learning method to make feature recommendations during the domain analysis process. Our feature mining and feature recommendation algorithms are quantitatively evaluated and the results are presented. Also, the performance of the recommender system is illustrated and evaluated within the context of a case study for an enterprise level collaborative software suite. The results clearly highlight the benefits and limitations of our approach, as well as the necessary preconditions for its success.

Index Terms—domain analysis, recommender systems, clustering, association rule mining, k-nearest neighbor.

1 Introduction

Domain analysis is the process of identifying, organizing, analyzing, and modeling features common to a particular domain [1], [2]. It is conducted in early phases of the software development life-cycle in order to generate ideas for a product, discover commonalities and variants within a domain, help project stakeholders configure product lines, and to identify opportunities for reuse. As such, it is an important component of the upfront software engineering process. In current practice, most domain analysis techniques, such as Feature Oriented Domain Analysis (FODA) [2] or Feature-Oriented Reuse Method (FORM) [3] rely upon analysts manually reviewing existing requirement specifications or competitors' product brochures and Web sites, and are therefore quite labor intensive. The success of these approaches is dependent upon the availability of relevant documents and/or access to existing project repositories, as well as the knowledge and expertise of the domain analyst. Other approaches such as the Domain Analysis and Reuse Environment (DARE) [4] utilize data mining and information retrieval methods to provide automated support for feature identification and extraction, but tend to focus their efforts on only a small handful of requirements specifications. The extracted features are therefore limited by the scope of the available specifications.

In this paper, we address these limitations through presenting a novel approach for identifying a broader set of candidate features. 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 partial product specifications found on Web sites which host or market software packages. It then analyzes the relationships between features and products and utilizes this information to recommend features for a specific project. Our approach takes as input an initial product description, analyzes this description, and then generates related feature recommendations utilizing two different algorithms. The first algorithm uses the unsupervised association rule mining [5], [6] technique to discover affinities among features across products and to augment and enrich the initial product profile. The second algorithm acts upon the augmented product profile and uses a k-nearest neighbor approach, common in collaborative filtering recommender systems [7], to analyze neighborhoods of similar products in order to identify new features. The end result is a set of recommended features which can be fed into the requirements engineering process in order to help project stakeholders define the features for a specific software product or product line. Figure 1 illustrates a feature recommendation scenario for an anti-virus

• *N. Hariri, M.Mirakhorli, J. Cleland-Huang and B.Mobasher are with the School of Computing, DePaul University, IL, 60604.*
C. Castro-Herrera is with Google Inc.
E-mail: nhariri@cs.depaul.edu, jhuang@cs.depaul.edu

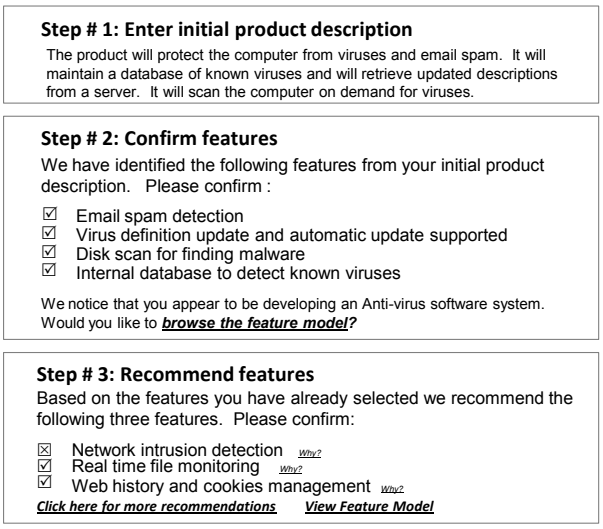
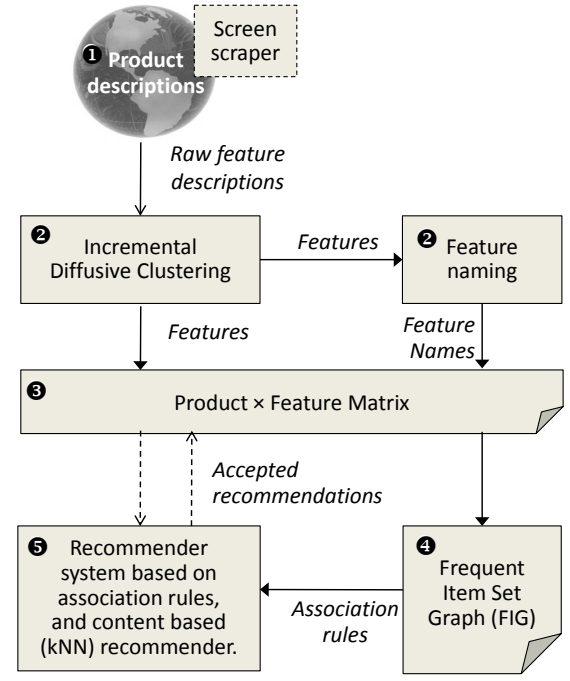


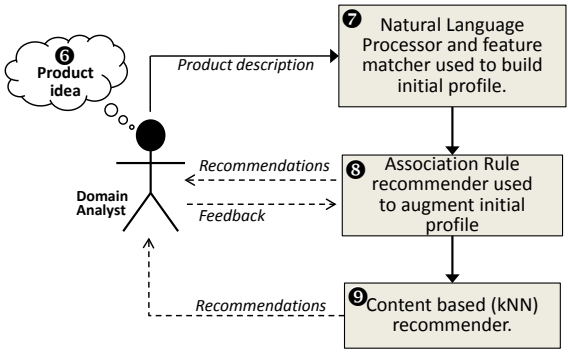
Fig. 1. Example of Feature Recommendations

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. This initial profile is then augmented to include additional features of network intrusion, file monitoring, and web history and cookies management. In our approach, the user can interact with the system in order to provide feedback on candidate features. In this example scenario, the user rejects *network intrusion*; however *file monitoring* and *web history* are added to the initial product profile. Additional recommendations (not shown) are then generated based on this profile.

In our previous work [8] we introduced our feature recommender system and evaluated it in a restricted domain with relatively small sized software products such as anti-virus software, multi-media iPod applications, and photography applications. In this paper, we extend this prior work by making several additional contributions. First, we provide a more detailed analysis of our technique for extracting features from online listings utilizing our Incremental Diffusive Clustering (IDC) algorithm. Previously, we had shown anecdotally that IDC performed well for clustering requirements and features. In this paper we present a quantitative evaluation of IDC by comparing it against well established algorithms including K-Means [9], Fuzzy K-Means [10], Spherical K-Means [11], and Latent Dirilecht Allocation (LDA) [12]; and we demonstrate that it outperforms them for purposes of clustering feature descriptors. Secondly, we demonstrate and evaluate our approach across a broader set of domains including applications for Internet, security, system solutions, office-tools, and programming domains. Finally, as our recommender system is intended to support domain analysis in large projects, we evaluate it within the context of a project designed to deliver



(a) Constructing the recommender system



(b) Using the recommender system to recommend features

Fig. 2. Feature Recommendations

a software suite for supporting remote collaboration. This project incorporates features from a broad set of domains and demonstrates the usefulness of our recommender system in a more realistic development scenario.

Our Feature Recommender System includes several different components as illustrated in Figure 2(a). The first component is the *Screen scraper* ① , responsible for mining *descriptors* from online product specifications. The second component is the *incremental diffusive clustering* algorithm ②, which clusters descriptors into *features* and extracts meaningful names for each cluster. The third component is a product-by-feature matrix ③, which is generated as a by-product of the clustering process and is then used to construct a *frequent itemset graph* (FIG) ④ [13], [14] from which feature *association*

rules can be generated. The final component is the recommender system itself ⑤. This system utilizes both the product-by-feature matrix and the FIG to generate on-demand feature recommendations. Figure 2(b) depicts how feature recommendations are made through a series of interactions with the user (steps ⑥ to ⑨).

The remainder of this paper is laid out as follows. Section 2 describes our approach for mining features from online product descriptions, provides a qualitative evaluation of the mined features, and reports the results of a comparative evaluation of IDC versus other well-established clustering techniques. Section 3 describes our feature recommender system, which is then evaluated in Section 4. Section 5 describes the case study we conducted in which the recommender system was used to support domain analysis activities for a Collaborative Online Software Suite. Section 6 discusses threats to validity for our work, section 7 describes related work, and finally section 8 summarizes our main findings and suggests areas of future work.

2 Feature Extraction

Many different websites such as SoftPedia, Google Apps MarketPlace, and Softonic, provide descriptions of software products. For purposes of our study, we utilized Softpedia.com, which indexes hundreds of thousands of software products. Most products in Softpedia include a product summary and a bulleted list of features. The Screen-Scraper utility was used to extract both the product summary and also the list of features. The summary data was split into sentences which were added to the features extracted from the bulleted list, creating a combined list which we refer to as *feature descriptors*. In this study, 493,347 feature descriptors were retrieved from 117,265 products, categorized under 21 categories and 159 sub categories. Table 1 depicts a sample of the feature descriptors mined from two typical SoftPedia products.

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 among features. For example, 80% of products containing parental controls also track web history, while 77% of Anti-Virus products that provide integration with 3rd party software do NOT support protected files.

TABLE 1
Sample Descriptors from Two SoftPedia Products

FAR 3.0

Drag and drop facility for copy and move operations;
Easy configurable options: internal/external file viewer and text editor, file operation associations for certain file types, panel view and file sorting modes;
Long file name support;
NTFS ”compressed” and ”encrypted” (Win2K) attribute and Hard/symbolic links support;
Plugin modules and commands: default plugins set includes Archive management plugin, FTP client, network browser, print manager and temporary panel, but you may write your own plugins;
Tunable configuration, color scheme customization;
Clipboard functions;

Windows Phone Device Manager

Applications management: view, install/uninstall homebrew applications;
File management: explore device, exchange files with your phone;
Send to Windows Phone (to send files, apps, ringtones, web links in one click);
Add and manage custom ringtones;
Send SMS, E-mail, notes directly from PC (without needing cloud services);
Take screenshots of the phone;
Programming sends notifications to phone;
Shared clipboard with PC and phone;
Integration with Windows explorer (drag & drop, copy/paste, view apps icons and details);

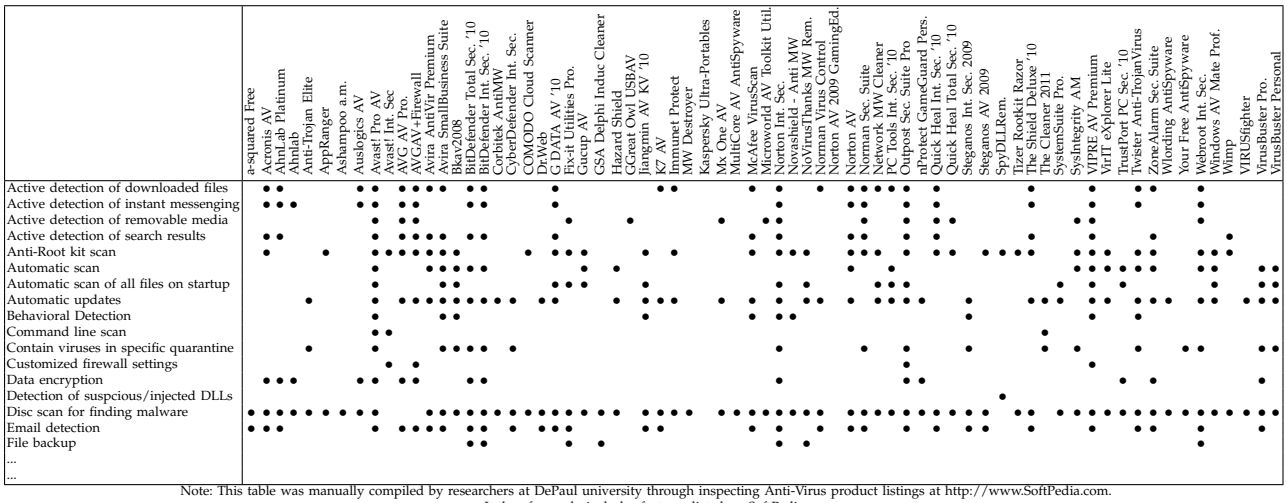
2.1 Incremental Diffusive Clustering

The center piece of the feature discovery component of the system is our Incremental Diffusive Clustering algorithm [8], [15], [16]. The goal of this component is to group together extracted product feature descriptors into an aggregate representation of a candidate feature. Thus, each discovered cluster will represent a feature that captures common characteristics across products. We originally developed IDC to address perceived cohesiveness problems observed when clustering short documents such as requirements, features, and fault descriptions, using existing clustering algorithms [17].

Although anecdotally IDC outperformed other techniques for clustering features, we did not previously evaluate it in a rigorous manner. In this section, we first describe the IDC algorithm, and then report on a new quantitative and qualitative analysis that we conducted.

IDC is an incremental clustering approach, in which clustering is achieved through a series of iterations. The best cluster is retained from each iteration, and its dominant terms are identified and then removed from all feature descriptors. This allows cross-cutting, yet non-dominant, topics to emerge in subsequent clustering iterations, resulting in fuzzy clusters in which each feature descriptor can be assigned to more than one cluster [8], [18]. The iterative diffusion process is important in obtaining clusters that both identify unique

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.

characteristics that contribute to a given feature, as well as retain those characteristics that are common across distinct features. Our goal in designing the IDC algorithms was to be able to generate overlapping clusters that correspond to product features, yet at the same time maintain the clustering cohesiveness that is typically achieved by crisp clustering techniques, such as standard variations of k-means algorithm. Our experimental evaluation, presented later in this section, shows that the IDC algorithm does indeed achieve this goal.

2.1.1 Preprocessing

Initial feature descriptors are first preprocessed by tokenizing them into a set of keywords, removing commonly occurring words (stop words), and stemming the remaining keywords to their root form (i.e., terms). Given the dictionary of all such terms $T = \{t_1, t_2, \ldots, t_W\}$, each feature descriptor, 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 feature descriptor f_i. These term weights are then transformed using a standard term-frequency, inverse document frequency (tf-idf) approach [9] such that, $tf\text{-}idf(f_{i,j}) = f_{i,j} \cdot log_2(D/df_j)$ where D represents the total number of feature descriptors, and df_j represents the number of feature descriptors 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.

2.1.2 Granularity

To determine how many features (clusters) to generate for a given product category, IDC uses a modified version of Can's metric [19] which considers the degree to which each feature descriptor differentiates itself from other feature descriptors. 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 were not relevant or useful for identifying potential features.

2.1.3 Clustering

The feature descriptor vectors extracted in the preprocessing stage are automatically clustered using the incremental diffusive clustering algorithm (IDC) [8], [15] in order to extract a meaningful and cohesive set of final features. In each iteration, IDC first automatically clusters the feature descriptors, and then, it identifies and retains the highest quality cluster based on the cohesiveness and size of the cluster. Once this cluster has been selected, its dominant terms are identified and removed from all clusters. This allows other cross-cutting, but non-dominant, topics to emerge in the subsequent clustering iterations. In our initial rendition of IDC, a consensus-based Spherical K-Means clustering approach was used in each iteration in order to increase the cohesiveness of intermediate clusters. However, to address scalability issues associated with clustering 493,347 feature descriptors, we replaced consensus clustering with the use of standard Spherical K-Means clustering. Furthermore, to reduce runtime, we clustered feature descriptors in

each of the Softpedia categories separately, and then merged similar clusters across the categories.

2.1.4 Selecting the best cluster

In each iteration IDC 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 topics. 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 feature descriptors 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. Note that if cohesion alone had been used to select the "best" cluster, the algorithm would have been biased towards small clusters. In the extreme case, a cluster with only one feature descriptor would always be selected. Topic coverage was therefore used as a second criterion to balance cohesion and cluster size.

2.1.5 Removing dominant terms

To remove dominant terms, terms exhibiting weights above a predefined threshold (0.15) in the centroid vector 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*, *email* and *encrypt*, 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.

This process is repeated until the targeted number of features, determined previously in Step 1, have been identified.

2.1.6 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 feature descriptor in the corresponding cluster. Descriptors whose similarity to the centroid fall below a given threshold value (set to 0.35 based on empirical observations) are removed from the cluster. Centroids are then repositioned according to the remaining feature descriptors using the same SPK algorithm adopted in the initial clustering step. Missing feature descriptors are identified by recomputing the cosine similarity between every feature descriptor not currently assigned to a cluster, and the centroid of that cluster. Any feature descriptor exhibiting a similarity score higher than a given threshold (set to 0.35) is added to that cluster. Finally, similar clusters are merged by computing the cosine similarity between each pair of centroids. Any pair of clusters exhibiting a score above a given threshold (set to 0.55) is finally merged into a single feature.

2.1.7 Feature naming

Each feature is named by 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 feature descriptor and the centroid of the cluster, and then averaging all term weights in the feature descriptor vector above a certain threshold (0.1). The cosine similarity and the average of dominant term weights are then added together to produce the score of each feature descriptor. The feature descriptor scoring the highest value is selected as the medoid and the corresponding original feature descriptor (from Softpedia) is selected as the name for the feature. 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*.

2.1.8 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.

2.2 Quantitative Cluster Evaluation

Because of the importance of the feature mining process in our system, we performed a quantitative analysis of the generated features. Quantitative cluster evaluation measures are generally classified into two types: unsupervised and supervised. Unsupervised measures such as cluster cohesion and cluster isolation, evaluate the clustering algorithm without reference to external information, and therefore fail to evaluate how well the generated clusters support specific tasks [9]. They also tend to measure localized cluster quality rather than optimizing for a global utility. In contrast, supervised measures, often referred to as *external measures*, provide a more general assessment of quality by evaluating the extent to which generated clusters match an external ideal structure or ground truth (usually provided in an answer set). For our evaluations, the ideal clusters, also called *classes*,

were produced by human experts and with manually clustering feature descriptors.

IDC was evaluated in the context of three different datasets (i) *antivirus* product descriptions extracted from SoftPedia, (ii) *multimedia video* product descriptions extracted from SoftPedia, and (iii) a set of 809 feature requests which were collected from approximately 30 people for an Airport Kiosk in a requirements management forum. For the first two datasets, clusters were formed by members of our research team. To accomplish this, feature descriptors in each category were first pruned to remove redundancies. After pruning the antivirus dataset, 830 of 1075 features remained which were then grouped into 27 clusters. Similarly, the set of 3710 features in the multimedia dataset were reduced to 3176 features and were clustered into 25 groups by experts. For the *airport* dataset, an ideal clustering was produced by a team of 4 college students who conducted a card sorting exercise [20] which produced 40 groups.

For evaluation purposes four different competing approaches were selected. These included Latent Dirichlet allocation (LDA) [12], Spherical K-Means clustering [11], K-Means, and Fuzzy K-Means [10] clustering algorithms. K-Means algorithm is a common baseline to compare and evaluate clustering algorithms and is also used in our experiments as one of the competing methods. To determine how much the variation of the Spherical K-Means clustering used in the IDC approach can improve the standard algorithm, we compared the results of our algorithm with the Spherical K-Means method.

The clusters produced by both K-means and Spherical K-Means algorithms are *crisp* clusters that do not contain overlaps. In our experiments, Fuzzy K-means was selected as a competing clustering technique which produces *soft* clusters where each item can have a degree of membership in each cluster.

Instead of using vector space models to represent feature descriptors as in K-Means and IDC, there are other techniques that find the latent topics in each descriptor. Some well-known techniques in this category are the Probabilistic Latent Semantic Analysis (pLSA), the Latent Dirichlet Allocation (LDA), and the Mixture of Dirichlet Compound Multinomial (DCM). Among these models, we selected LDA to be used in our comparisons as it has been shown to perform well for semi-structured and text-oriented data.

2.2.1 Purity

Purity is measured by comparing generated clusters to the answer set clusters (also called classes). Each generated cluster is matched with the answer set cluster with which it shares the most descriptors. Purity is then computed by counting the number of correctly assigned descriptors and dividing by the total number of descriptors shown as N. In other words, let $W = \{w_1, w_2, .., w_n\}$ be the set of clusters found by the clustering algorithm and $C = \{c_1, .., c_l\}$ be the set of classes. Purity can be computed as follows [9]:

$$purity(W, C) = \frac{1}{N} \sum_k max_j |w_k \cap c_j| \qquad (2)$$

Purity can take values in the range of 0 to 1. A perfect clustering algorithm has a purity of close to 1 while a poorly performing clustering method could have purity closer to 0.

Fuzzy K-Means and LDA are soft clustering methods and produce membership probabilities for each descriptor. There are two approaches for comparing these methods against crisp clustering algorithms like K-Means and Spherical K-Means; one way is to place a threshold on the membership probabilities and then for each descriptor to choose all clusters for which it has probabilities above the threshold. The second approach is to only select the highest probability cluster for each descriptor. The quality of the clustering method also depends on the parameters and threshold values used in each of the methods. For experimental purposes, we ran each technique with different parameters and then used the best result for comparison against other techniques. All four algorithms were configured to produce the same number of clusters as the answer set.

As noted earlier, Purity is a localized metric and does not measure the global quality of the clustering as a whole. Generally, for larger numbers of clusters, each cluster will exhibit higher degrees of purity. In the extreme case where the number of clusters is equal to the number of items, purity is maximized at 1.0. To mitigate this problem, we also computed normalized mutual information (NMI).

2.2.2 Normalized Mutual Information (NMI)

NMI is calculated as follows:

$$NMI(W, C) = \frac{I(W; C)}{[H(W) + H(C)]/2} \qquad (3)$$

In the above formula, $I(W; C)$ represents mutual information and is computed as in equation 4. This value is a measure of the amount of information that clusters contain about classes. Mutual information is zero if the clusters are totally unrelated to actual features.

From the NMI formula, H represents entropy which measures the degree to which each cluster consists of objects of a single category.

$$I(W; C) = \sum_k \sum_j \frac{|w_k \cap c_j|}{N} log \frac{N|w_k \cap c_j|}{|w_k||c_j|} \qquad (4)$$

$$H(W) = -\sum_k \frac{|w_k|}{N} log \frac{|w_k|}{N} \qquad (5)$$

2.2.3 Analysis of the results

As discussed in section 2, feature descriptors are generated by splitting the product summary data into sentences and taking each sentence as a descriptor. There can be situations where a sentence describes more than one feature. The problem with using crisp clustering techniques, such as Spherical K-Means and K-Means, is that some descriptors, common across multiple features, may be assigned to only one feature. The IDC clustering algorithm, however, produces overlapping clusters in which each feature descriptor can be assigned to more than one cluster. Although this aspect of the clustering algorithm is not captured by the common evaluation metrics including purity and NMI, it is particularly important for our feature recommendation system. As an example, consider Table 3 that shows two overlapping clusters generated by the IDC algorithm. In this example, one of the clusters relates to the *Joining multiple video files* feature while the other cluster represents *Splitting a large video file* feature. A representative set of descriptors for each of the two clusters, as well as a sample of common feature descriptors, are presented. To illustrate the problem addressed here consider the feature descriptor, *Split and join screen recordings*, that describes two separate features (split, and join). By applying crisp clustering methods, this descriptor is assigned to one, and only one, of the two related features; while the IDC method assigns it to both clusters.

As noted earlier, our goal in designing the IDC algorithms was to be able to generate the type of overlapping clusters described above, and yet maintain the clustering effectiveness (as measured by the NMI and purity metrics) that is typically achieved by crisp clustering techniques. Although LDA and Fuzzy K-Means methods can also produce soft clusters, IDC generally showed better performance in terms of purity and NMI in comparison to these two methods and therefore was used in our approach.

Figure 3 compares the purity of IDC with other clustering methods. For the antivirus dataset, Spherical K-Means achieved slightly better purity than IDC (3% improvement) while other competing methods returned lower purity than IDC. For the multimedia video dataset, IDC had the best performance and improved Spherical K-Means and standard K-Means algorithms by 11% and 16% respectively. For the Airport dataset Spherical K-Means, K-Means, and IDC all had very similar purity.

Figure 4 depicts the NMI for different clustering methods for each of the three datasets. IDC and Spherical K-Means algorithms both returned higher NMI than the other competing methods over all the three datasets. Similar to the results for purity, IDC and Spherical K-Means had very similar performances in terms of NMI. More specifically, Spherical K-Means returned about 1.5% higher NMI for the antivirus dataset while IDC returned 1% higher NMI for multimedia video dataset. These two methods had the same NMI for the airport dataset.

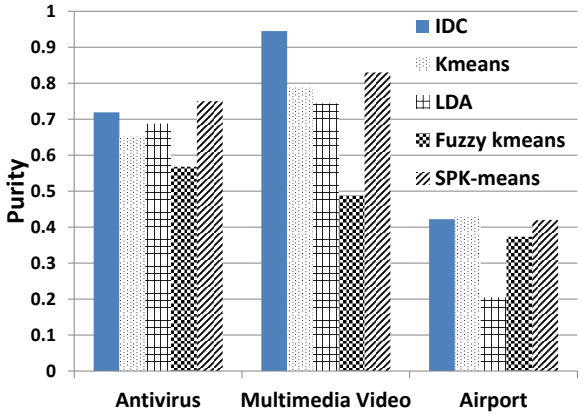


Fig. 3. Comparison of Purity for Different Clustering Algorithms

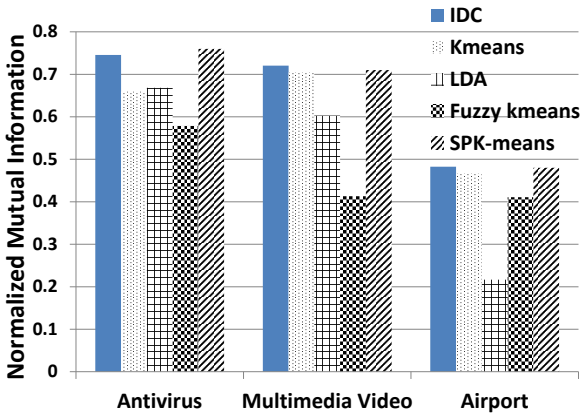


Fig. 4. Comparison of Normalized Mutual Information for Different Clustering Algorithms

2.2.4 Recall and Precision

As explained in section 2.2.3, in some cases a feature descriptor describes more than one feature. If crisp clustering methods are used, valid features can be missed for a descriptor, and as a result, the generated product-by-feature matrix is unnecessarily sparse. This is undesirable, as sparsity is a major issue known to limit the quality of recommendations [21][22].

Based on our evaluations in the previous sections, IDC and SPK-Means had very similar performances in terms of purity and NMI (while better performance in comparison to the other baselines). In this section, we compare the precision and recall of assigning descriptors to features for these two clustering methods. This evaluation was performed by three students who were familiar with concepts of software engineer-

TABLE 3
An Example of Two Overlapping Clusters

Cluster A - Join Multiple Files into One Large File	**Cluster B - Split a Large File into Smaller Files**
·Join multiple video files into one large AVI, MPEG, WMV file ·Agile MPEG Video Joiner is a powerful easy-to-use tool to join multiple video files ·It supports joining multiple MPEG, AVI-DivX, WMV-ASF and other video file formats ·Join/merge MPEG-1 and MPEG-2 video	·Split, cut a large video file into smaller video,audio clips ·VidSplitter is a complete Video splitting tool that lets you to split large MPEG, ASF files into smaller video clips in various formats · Split video clips out from MPEG4, MP4, MPEG easily with only 3 steps ·Split file between all supported formats
Sample of Common Feature Descriptors	
· MP3 MPEG Joiner/Splitter is a software that joins/splits files of the type .mp3 or .mpg · Split Join Convert MOV is an efficient program designed to split, join or convert Quicktime MOV or QT files to AVI, MPEG or WMV formats · Split Join Convert Video is a versatile tool to split, join or convert video files · Split and join screen recordings · Video Splitter Joiner and Converter is an all-in-one tool to split, join or convert video files · A handy video file splitting and joining discount pack that includes Ultra Video Splitter and Ultra Video Joiner	

ing and domain analysis. However, the evaluators did not have any prior knowledge of our data and the specifics of our project. For this evaluation, a sample of 32 feature descriptors were selected from the antivirus and multimedia software categories of Softpedia. The sample was selected such that half of the descriptors were assigned to more than one cluster based on IDC while the remaining descriptors belonged to only one cluster. The reason for this selection was to evaluate the false positive mappings (a descriptor is assigned to a feature while it should not be) and true negative mappings (a descriptor should be assigned to a feature but it is not).

The users were presented with the union of the features discovered by IDC and SPK-Means. For each descriptor, the participants were asked two questions: (1) *Name the set of features that should be assigned to this descriptor.* (2) *If any, name the set of features that should be assigned to this descriptor but are not in the set of presented features*.

For each descriptor, recall and precision were computed based on the answers from each participant and were then averaged to compute the final recall and precision for that descriptor. As the number of test items is more than 30, we can assume that the sample is normal. We used the Student's t-test to evaluate the following two null hypotheses:

H_p: The mean precision achieved using IDC is equal to the mean precision achieved using SPK-Means.

H_r: The mean recall achieved using SPK-Means is greater than or equal to the mean recall achieved using IDC.

Table 4 presents the average precision and recall for IDC and SPK-Means as well as the p-value for the two tests. The results show that while the mean precision for IDC and SPK-Means are not significantly different, the mean recall for IDC is significantly higher than SPK-Means.

TABLE 4
Student's t-test results for comparison of mean precision and mean recall for IDC and SPK-Means

	Average for SPK-Means	Average for IDC	p-value
H_p	0.85	0.82525	0.632484247
H_r	0.532142857	0.680059524	0.000906678

2.3 Qualitative Feature Evaluation

We also performed a qualitative evaluation of the clusters generated by each technique. This was particularly important because our recommender system will be used to support domain analysis tasks and so must generate clusters which are meaningful and understandable to human users. To assess the quality of the generated features, the three product categories of *antivirus* software, *multimedia iPod* applications, and *Photography* applications, were evaluated. All descriptors from each product in these categories were retrieved from the SoftPedia Web site. A card-sorting activity was then conducted to group feature descriptors into features. This task was performed and validated by three students who were familiar with these domains, but otherwise previously unrelated to this project. These manually identified features served as the answer set for the remainder of the evaluation. Each answer set took between 18-25 hours to construct, and resulted in the identification of 27 distinct features for antivirus, 30 for iPod, and 38 for multimedia photography.

In the next step of the evaluation, the IDC algorithm was used to automatically identify features. These features were then evaluated against the answer sets by three additional students at DePaul. 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 5 and show that while our approach discovered only about 56% of the features in the answer set, it also discovered about 30% new unique features that were missed in the manually constructed answer set. This occurred despite the fact that analysts spent from 18-25 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 descriptors beyond items listed as features. It is also worth noting that our naming algorithm performed very well, delivering almost 85% of features there were categorized as Good or Excellent. Only 2% of names were deemed inadequate.

TABLE 5
User Evaluation of Feature Quality and Completeness

	Virus	iPod	Photo	Overall
Total features in answer set	27	30	38	95
In both auto gen. and answer set	18	15	20	53
Answer set only	9	15	18	42
Auto generated only	9	15	19	43
Redundant features	3	3	1	7
Ill-formed features	4	2	2	8
Excellent name	0.45	0.50	0.69	0.57
Good name	0.20	0.36	0.27	0.28
Poor name	0.25	0.11	0.04	0.13
Inadequate name	0.05	0.03	0.00	0.02

3 Feature Recommendation

Based upon the clusters generated using IDC, we create a binary product-by-feature matrix, $M := (m_{i,j})_{P \times F}$, where P represents the number of products (117,265), F is the number of identified features (1,135), and $m_{i,j}$ is 1 if and only if the feature j includes a descriptor originally mined from the product i. This matrix, which contains the complete set of recommendable features referred to as the *feature pool* from now on, is used to generate feature recommendations in the following steps.

3.1 Creating initial product profile

First an initial product profile is constructed in a format compatible with the feature model. To accomplish this, the domain analyst creates a short textual description of the product, which is then processed in order to match elements of the description to features in the feature pool. This matching is accomplished by first performing basic preprocessing, such as tokenization, stemming, and removal of stop words, and then converting the product description p to a term vector $p = (w_{1,p}, w_{2,p}, ..., w_{n,p})$ where each dimension corresponds to a separate term. We used the standard *term frequency-inverse document frequency* approach [9], also known as $tf - idf$, which computes the weight for each term based on the *normalized term-frequency* and the *inverse document frequency*.

Once the product description is converted to term vector form, it is compared to the term vector representation of each feature in the feature pool using a standard information retrieval metric such as the cosine similarity. Features are then ranked according to their similarity to the product description, and presented to the analyst for confirmation. In certain cases, the product description might describe a feature which is not matched to the feature pool. This may occur because the feature pool does not provide coverage of the intended functionality. In this case, the recommender system will be unable to incorporate this information into the recommendation process. In other cases, a match may not be found because the analyst did not describe the product in sufficient detail or used wording not represented in the model. In this case, the analyst can expand the description in an effort to make a match. To implement this phase of the process we utilized Poirot [23], which is a tool originally developed by our research group to support automated trace retrieval. Its search capabilities were ideally suited to the feature matching task.

As a result of this step of the process, an initial product profile is appended to the product-by-feature matrix.

3.2 Feature Recommendations using Association Rule Mining

As previously explained, our approach utilizes two different recommendation algorithms. The first algorithm addresses the potential problem that the initial profile is relatively sparse and contains only a few features. To rectify this problem, we utilize Association Rule Mining [5] which is known to return relatively accurate, albeit incomplete, recommendations. It is therefore ideal for the first step of augmenting the product profile.

Association Rule Mining was originally developed to support "market basket analysis" in order to analyze products that buyers tend to purchase at the same time. For example, an association rule $milk, butter \implies bread$ means that customers who purchase milk and butter are also likely to purchase bread. Association rule mining has been used to build recommender systems in several different domains including e-commerce and intelligent Web applications [24], [13], [25]. Association rules can be learned

from the product-by-feature matrix, and then used to make recommendations. For example, consider a rule stating that *email spam detector, virus definition update* $\Longrightarrow$ *web history and cookies management*, then a product profile containing an email spam detector and a virus definition update, could result in a recommendation to add a feature to support web history and cookies management. In general, when a partial profile is matched against the antecedent of a discovered rule, the items on the right hand side of the matching rules are sorted according to the confidence values for the rule, and the top ranked items from this list form the recommendation set.

More formally, given a set of product profiles P and a set of features $F = \{F_1, F_2, \cdots, F_k\}$ and a feature set $f \subseteq F$, let $P_f \subseteq P$ be the set of products that have all the features in f. The *support* of the feature set f is defined as $\sigma(f) := |P_f| / |P|$. 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 $X \implies Y(\sigma_r, \alpha_r)$, where $X \subseteq F$ and $Y \subseteq F$ are feature sets, σ_r is the *support* of the rule r defined as $\sigma_r := \sigma(X \cup Y)$, and α_r is the *confidence* for the rule r given by $\alpha_r := \sigma(X \cup Y)/\sigma(X))$. The discovery of association rules involves two main parts: the discovery of frequent feature sets (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. Various algorithms exist for discovering the frequent itemsets and association rules among which we chose to use FP-Growth algorithm [26] as it is shown to be quite memory-efficient and hence suitable for the size of our data set.

After creating the association rules, new features can be recommended to an initial product profile by finding all the matching rules. In order to reduce the search time, the frequent feature sets are stored in a directed acyclic graph, called a Frequent Itemset Graph (FIG)[13], [14]. 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 feature sets of size $d + 1$ that contain I at the next level. The root node at level 0 corresponds to the empty feature set. Each node also stores the support value of the corresponding frequent feature set.

Given a product profile with feature set f, a depth-first search is performed on the graph to level $|f|$. If a match is found, then the children of the matching node are used to generate candidate recommendations; and each child of this node corresponds to a frequent itemset $f \cup \{r\}$. A feature r will be added to the recommendation list if the support ratio $\sigma(f \cup \{r\})/\sigma\{f\}$ exceeds a pre-specified minimum threshold. The same procedure will be repeated for all the subsets of the feature set f.

Figure 5 depicts a section of the graph for features related to *audio/video format conversion*. Each node represents a frequent feature set and the number associated with it shows the support value of the feature set. For example, the three features (i) *Supports conversion between popular audio files*, (ii) *Fast conversion speed and excellent output quality* and (iii) *Supports batch conversion*, occur together in 0.005 of total products and are recognized as a frequent feature set.

Features recommended by the association rule mining approach are then presented to the user. The user selects the correct recommendations and these features are used to augment the initial product profile. The augmented profile is then given as input to the kNN recommendation module to produce more recommendations.

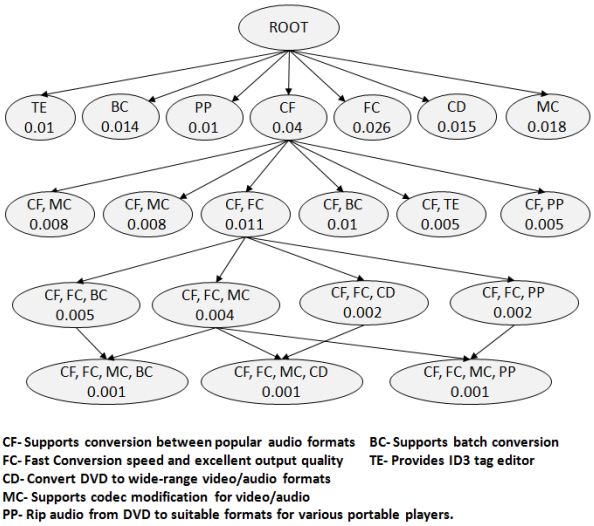


Fig. 5. Part of a Frequent Itemset Graph for the *audio/video format conversion*

3.3 Feature Recommendations using Binary kNN

In our prior work, the product-based k-Nearest Neighbor (kNN) algorithm has been shown to be an efficient method for recommending features and requirements [27], [28]. For purposes of feature recommendations, the similarity of the new product and each of the existing products in the product-by-feature matrix is computed and the top k (25) most similar products are selected as *neighbors* of the new product (note that products with less than 6 features were ignored, as their profiles are too sparse to create good neighborhoods). Then, the binary equivalent of cosine similarity is used to compute the similarity of the new product p and the existing product n, productSim(p, n), as follows:

$$\text{productSim}(p,n) = \frac{|F_p \cap F_n|}{\sqrt{|F_p| \cdot |F_n|}} \quad (6)$$

where F_p denotes the set of features of product p [29]. After forming the neighborhoods, features can be recommended to the new product using a similar approach as in [7]:

$$\text{pred}(p,f) = \frac{\sum_{n \in nbr(p)} \text{productSim}(p,n) \cdot m_{n,f}}{\sum_{n \in nbr(p)} \text{productSim}(p,n)} \quad (7)$$

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 top R features with highest predictions will be selected.

To further reduce the noise in the recommendations, a post-filtering step is added to the recommendation process to re-rank the recommendations in favor of the domain-specific features. In order to accomplish this, a *cross-cutting factor* is computed for every feature by counting the number of product types that are covered in the corresponding cluster. The prediction score given by the kNN method for each feature is then divided by its cross-cutting factor to produce new scores. The features are descendant ordered based on these new scores and the top N recommendations are presented to the users.

4 Feature Recommender 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 [30].

We adopted a standard experimental design based on a m-fold cross validation approach in which the data was divided into m sets. In each of the m runs, one of the sets is used as the test set and the union of the remaining $(m-1)$ sets are used for training the recommender system. After m runs of the experiment, each of the m sets was used once as a testing set. The performance of the system was determined by averaging its performance over the m runs.

The goal of these experiments is to determine the performance of the system in making correct feature recommendations for a product with a small set of known features. Therefore for each product in the test set, L features were randomly selected and used to represent the initial product profile. One of the remaining features was then randomly selected as a target item, and the recommender system was evaluated with respect to whether it was able to recommend back this targeted item. If the feature was recommended, its position in the overall recommendation list for the run was recorded. The rationale behind recording the rank of the recommendation is because top recommendations are more valuable to the users, who are less likely to scroll to the end of a long list of recommendations.

Results for leave-one-out cross validation recommendation experiments can be 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 $f_p \in R_N(p)$, a *hit* is recorded 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.

In practice, the usefulness of the recommendations degrades for lower ranks as the users do not usually scroll down to see all the recommendations. Therefore, to compare the performance of two competing algorithms, differences in higher ranks should be given more weight. Although hit-ratio ratio graphs provide a good illustration of the performance of the algorithms at different ranks, comparing the hit-level mean values will not capture this point.

In case of our leave-one-out cross validation experiment, for the test product p, if the target feature f_p is recommended at level X, then the precision of recommendations for p is $\frac{1}{X}$. we evaluate the significance of the differences between mean precision in our comparisons (instead of mean hit-level). This is mainly because average precision is less sensitive to differences in lower ranks than differences in higher ranks.

4.0.1 Experiment 1: Comparison of different recommendation algorithms

The first experiment was designed to evaluate the performance of the product-based kNN used in our rec-

ommendation module, against two well-known recommendation methods, namely, feature-based kNN [31] and matrix factorization using Bayesian personalized ranking (BPRMF) [32]. Similar to product-based kNN, feature-based kNN method is also a neighborhood model, but one which makes predictions based on feature neighborhoods. Matrix Factorization models are an alternative approach which map both products and features to a joint latent factor space of dimensionality q, such that product-feature interactions are modeled as inner products in that space. BPRMF, which was used in our experiments, is a form of matrix factorization method designed to be most useful for binary data.

A five-fold cross validation approach was followed to evaluate the performance of the recommendation methods. The experiment was performed with an initial profile size of $L = 3$. The number of neighbors for product-based kNN and feature-based kNN was set to $K = 20$. Also, the BPRMF method was run for 80 factors and in 3000 iterations with a learning rate of $\alpha = 0.05$.

The hit-ratio results of this experiment for a sample of Softpedia software categories are shown in Figure 6. For each category, the hit ratio is shown for different sizes of recommendation.

In all categories, the product-based kNN algorithm performed better than the feature-based kNN method for all confidence levels. Also, the graphs show that the product-based kNN outperformed BPRMF for the top 25 recommendations while the BPRMF has much better performance for larger sizes of recommendations. Assuming that it is not practical to show more than 20 results to the user at one time, the product-based kNN algorithm has the best performance.

We also computed the average precision of the competing algorithms and tested the significance of the differences between product-based KNN and the other two methods using the student's t-test. For each of the categories we tested the following two null hypotheses:

H_1: Mean precision for product-based kNN is less than or equal to the mean precision for feature-based kNN.

H_2: Mean precision for product-based kNN is less than or equal to the mean precision for BPRMF.

The t-test requires a normally-distributed sample. By the central limit theorem, sample means of moderately large samples (more than 30) are often well-approximated by a normal distribution even if the data are not normally distributed. As the number of test samples is larger than 30, we can assume that the sample is normal. The average precision(AP) for the recommendation algorithms as well as the p-values for the t-tests are included in table 6. For all the categories except multimedia the two null hypothesis H_1, and H_2 can be rejected with high confidence showing that product-based kNN can achieve a higher mean precision than the other two methods. For the multimedia category, although H_2 can be rejected, there is insufficient evidence to reject H_1.

4.0.2 Experiment 2: Association Rules

To determine the effect of association rule mining on the performance of the system, the previously described five-fold approach was adopted. In each of the five runs of the experiment, a frequent item set graph was constructed from the products in the training set. For each product in the test set, $L = 3$ features were selected and the remaining features were removed from the profile. The frequent item set graph was then used to generate recommendations, and those recommendations with confidence scores of 0.2 or higher, were presented to the user. Figure 7 shows the precision and recall for different levels of confidence. For example, at a confidence level of 0.2, the precision of the generated recommendations is 25% and recall is 37%. The association rule approach can therefore achieve high precision in recommendation, but at the cost of lower recall. These observations support our earlier claims that Association Rule Mining can be useful for identifying a small set of previously unused features with a high degree of precision.

To simulate the step in which the user evaluates the initial recommendations, we automatically accept the correct recommendations and reject the incorrect ones based on the known data stored in the product-by-features matrix. These accepted recommendations are then used to augment the initial product profile of size $L = 3$. The augmented profile is then given as input to the kNN recommender to generate more recommendations. In our evaluations, this hybrid method is shown as kNN+.

The leave-one-out cross validation experiment described in Experiment 1 was then conducted on the whole set of products, covering all of the categories, with the small modification that the left-out item was selected from the set of features that were not part of the augmented profile. Figure 8 compares the hit ratio results of the product-based kNN approach with the kNN+ method. As can be seen, the quality of recommendations can be improved when association rules are used to augment the product profile before running the kNN algorithm. This difference represents a 0.1 improvement in hit ratio at a rank of 20.

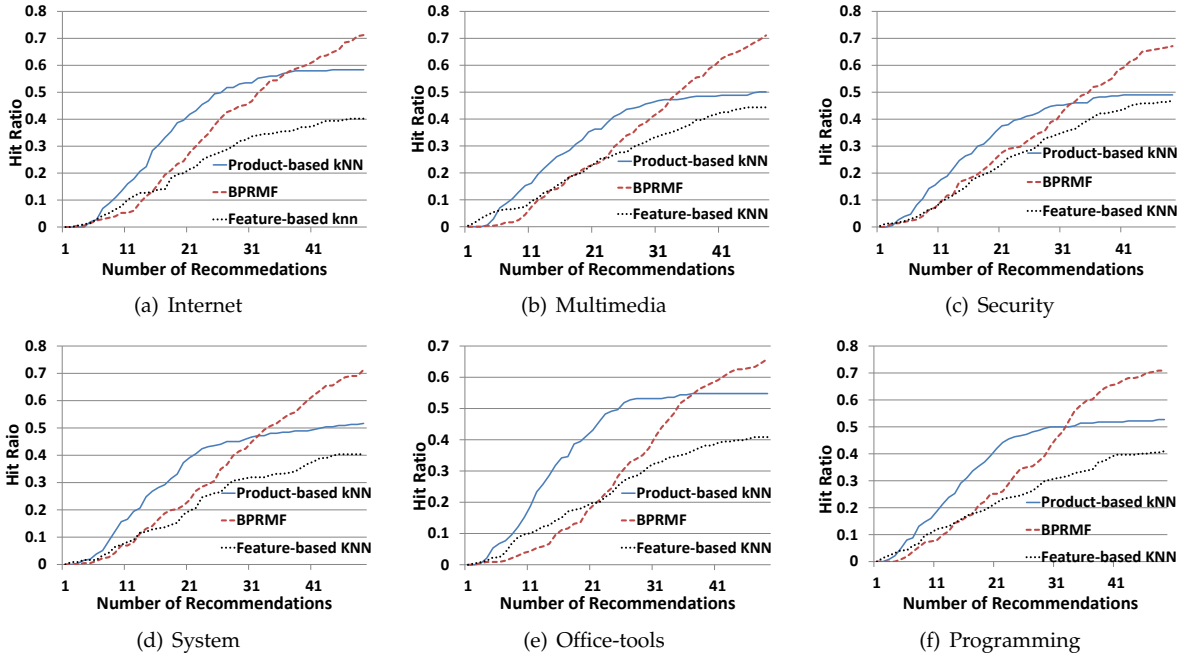
We also computed the average precision for these two methods. The average precision of kNN was 0.051 while the average precision of kNN+ was 0.060 . Similar to our evaluations in section 4.0.1, we used the Student's t-test statistic to test the following null hypothesis:

H_0: Mean precision for kNN+ is less than or equal to the mean precision for kNN.

The computed p-value for this test was 4.951E-49 which is much smaller than the commonly used

TABLE 6
Student's t-test results comparing the mean precision for product-based kNN and feature-based kNN and BPRMF for various categories

	AP for Product-based kNN	AP for Feature-based kNN	AP for BPRMF	p-value for H_1	p-value for H_2
Internet	0.044385257	0.029598626	0.038987167	1.22251E-16 *	0.003991202 *
Multimedia	0.042550062	0.040130303	0.037031697	0.287419169	0.026067424 *
Security	0.044682807	0.034942399	0.041098306	4.51717E-05 *	0.064780269 *
System	0.042300756	0.031335661	0.03758579	0.000200299 *	0.026554663 *
Office	0.049005664	0.031160901	0.035160479	3.7573E-17 *	8.83999E-06 *
Programming	0.049108903	0.036690594	0.04054354	8.07944E-06 *	0.002525262 *



(a) Internet (b) Multimedia (c) Security

(d) System (e) Office-tools (f) Programming

Fig. 6. Comparison of Recommendation Methods for Different Software Categories

threshold of 0.05. Therefore, we reject the null hypothesis.

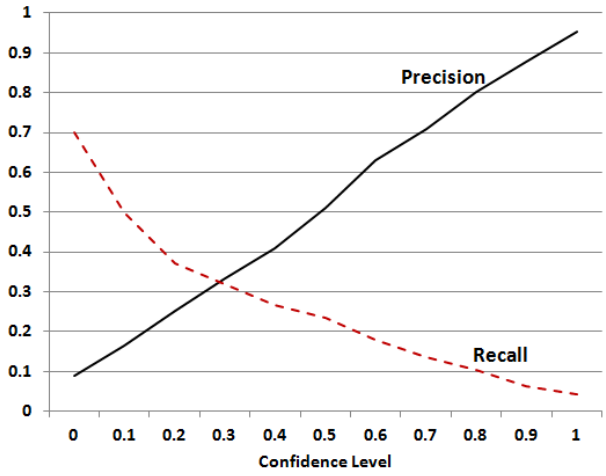


Fig. 7. Recall and Precision at Different Levels of Confidence

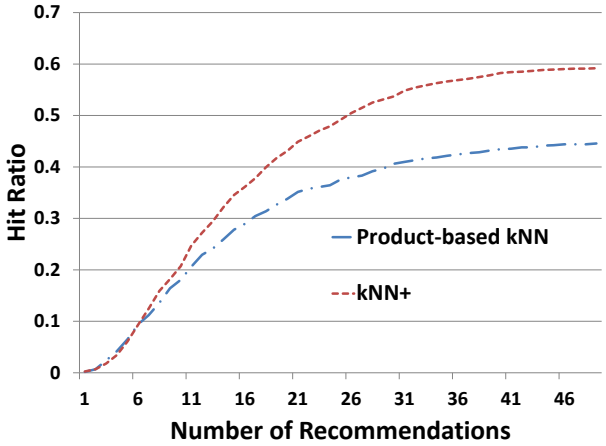


Fig. 8. Hit Ratio Comparison of kNN and kNN+

5 Case Study: Collaborative Software Suite

The previous experiments demonstrate the viability of utilizing our recommender system to recommend

useful features to a domain analyst. However, to further evaluate its usefulness we applied it within the context of a larger and more realistic project. We therefore present a case study, based around the Collaborative Software Suite (CoSS), which is intended to support project members collaborating on a distributed project. CoSS, will be developed as a web-based solution for managing projects and supporting collaboration across an enterprise-level organization. It will provide support for various activities including project management, virtual team management, advanced reporting, dashboards, online calendars, workflow creation, document management, communication and conferencing, etc.

5.1 Case Study Creation Methodology

Although different domains were considered, CoSS was selected because several researchers on our team were familiar with using enterprise level collaborative tools, and because the domain contained a relatively large number of interrelated features that were well represented in the SoftPedia data. The case study was developed by a member of our research team, from now on referred to as the Domain Analyst, who had extensive experience working on software development projects, and whose sole responsibility in this project was to build the case study and help in the qualitative evaluation of recommendations. The domain analyst followed the FODA method [2] and developed the domain model based on a combination of expert knowledge and publicly available product descriptions for over 100 related commercial products. To avoid introducing bias to the experiment, the analyst did not use Softpedia.com, which was the Website from which we mined features to train the recommender system. The domain analyst invested over 30 hours to identify approximately 120 coarse-grained features. The domain analysis did not include identifying composition rules such as optionality, variability, or mutual exclusion between features.

5.2 Feature Model Description

The CoSS feature model, depicted in Figure 9, shows primary features such as *project management* and *communication* and a selection of their associated sub-features. For example, *project management* is decomposed into features such as *project setup*, *task setup*, and *project notification*. Each feature is further defined through a textual description, as illustrated for *Task Setup* and its sub-features "Create new tasks, and to-do lists for each team member", "Create scheduled and recurring tasks" and "Create project/task workflow rules".

5.3 Qualitative Analysis

In addition to the quantitative study described in Section 4, we also conducted a qualitative evaluation which relied on expert human judgment. For this experiment, 5 tasks were defined where each task consists of a subset of 5 randomly selected features from the CoSS feature model and simulates a situation where an expert has selected 5 features for a product and is using the recommender system to discover more features. Table 7 shows one of the tasks used in our experiment.

As described in section 3.1, each of the feature descriptors are automatically mapped to the set of 1,135 features in our system to create an initial product profile for each task. The highest ranked matches are presented to the user for feedback. If no good matches are found, the user can modify the initial product description and reissue the recommendation request. Assuming that the feature that the user is looking for exists in our system, this process can continue until the user is satisfied that their initial product description is successfully matched to features. In our evaluations, it took less than 5 minutes in average to map each task to the set of corresponding features

In the next step, association rule mining was performed on the set of selected features and the results were shown to the user for feedback. Those confirmed recommendations were added to the profile. In the last step, the kNN algorithm was run and the top $R = 50$ recommendations were selected. These recommendations were then re-ranked based on the calculated cross-cutting factors and the top $N = 10$ features were shown to the users. Table 8 shows the set of recommendations produced by association rule mining and kNN algorithms for the sample task shown in Table 7. For each recommendation, the feature name and a sample of feature descriptors are provided.

An analysis of the recommended items was performed independently by three users who had not previously been exposed to the datasets and the COSS recommendations made by our approach. In order to remove bias in the experiment, for each of the tasks, the ranked list of N recommended features were mixed with N features that were randomly taken from the set of all features. A total of $2N$ recommendations were presented for each task. The users were not informed of the number of random features inserted in the recommendation list, and no differentiation was made between features generated by our method and the additional randomly selected ones. Study participants were asked to analyze each presented features in terms of the following criteria: (Q_1) *Is the recommended feature cohesive?* (Q_2) *Was this feature mentioned in the original profile?* (Q_3) *Is this feature related to the overall goals of the Collaborative Software Suite?* (Q_4) *Is this recommendation a possible feature for the Collaborative Software Suite?* and (Q_5) *Was this recommendation useful?*. It took less than 2 minutes in average for each user to answer this questions for each of the recommended features

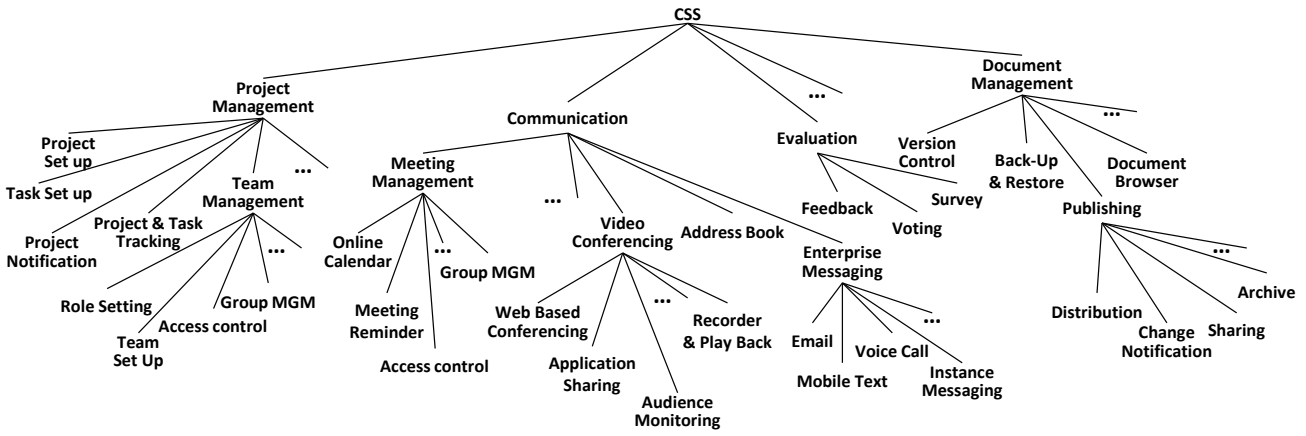


Fig. 9. Collaborative Software Suite: Domain Model.

TABLE 7
An Example of an Initial Product Description. For Experimental Purposes the Description Was Composed of a Set of Features Randomly Selected from the Manually Created Domain Model.

Provides different permission levels for adding and sharing calendars, events and so on.
Assign tasks to one or more of the team
Supports results reporting and charting
Provides data export import to excel, csv, spss file
Manage projects and teams and improve inter and intra-group communications

TABLE 8
Top Recommendations Showing Feature Titles and a Subset of Related Feature Descriptors

#	Algorithm	Feature Title	Sample Descriptors in the Feature
1	ARM	Extensive reporting, Customized reports	· Users can share customized reports easily. · Reporting: Save reports in Word Adobe HTML format. · Export reports in Excel files. · Custom reports generator.
2	ARM	Task manager	· Event and Task List · plan manage and schedule your important dates and times · Date Reminder for nonrecurring and recurring events
3	kNN	Features related to 'reminders'	· Flexible Reminder Frequency ·Built-in day planner to remind you of scheduled events ·Set Tasks and reminders with 1 click.
4	kNN	User Defined Filters	· Ability to filter report content by WorkItem type. · Filters for sorting items according to many factors. · Information is readily accessible in a variety of ways.
5	kNN	Templates	· Ready-to-use templates · Quickly enter your records using record templates · Quickly enter movie records using record templates
6	kNN	Features related to 'MS Outlook'	· Data can be synchronized with Microsoft Outlook · Duplicate Remover for Outlook Calendar · Outlook Express: Mail folders. · Outlook Express: Message rules.
7	kNN	Activity Tree Pane Features	·Activity Tree Pane Features: Provides a hierarchical arrangement of WorkItems · Activity Tree Pane Features: Ability to create your own WorkItem types. · Activity Tree Pane Features: Ability to track time spent on WorkItems.
8	kNN	Customizable categories for events, documents, tasks	· Filter Items by Category and Type. · Unlimited User defined categories. · Customizable Event Categories.
9	kNN	Features related to notes	· Export Notes ·Add a note to the event · It keeps track of your notes and tasks
10	kNN	Daily/weekly/monthly reporting or viewing activities	· The print options provide daily weekly and monthly summaries. · Weekly calendar view of your activities · Weekly View: Display events in a week.
11	kNN	Exporting to the Web	· Display record (text and graphic) in the form of the WEB page. · Publish your database to the Web · Web calendars.
12	kNN	Recording	· Screen Shot Capture is a handy and easy to use graphic capture tool · Screen Snapshot Recording · Video Recorder

For each task and for every user, the precision for each question from Q_1 to Q_5 was computed for the feature recommender and the random baseline. Given that we had 5 tasks and 3 evaluators, 15 paired samples were collected for each of the questions. Based on normality tests including Shapiro-Wilk test and also frequency histograms, we can assume that the samples were taken from a normal population.

For questions 2 to 5, the paired one-way Student's t-test statistic was used to evaluate the statistical significance of improvement of precision using our method. For question 1, we used paired two-way Student's t-test statistic to evaluate whether there was any difference in the cohesion of the features randomly generated and those produced by our recommender. Table 9 presents the null hypothesis for each question. In this table, μ_{rec} represents the mean precision for our recommendation approach and μ_{rand} indicates the mean precision for the random baseline. Also, the average precision over all the tasks and users are shown for our recommender and the random baseline.

According to Table 9, the p-values of the tests for questions Q_2 to Q_5 are all much smaller than 0.05 (which is commonly used as the threshold) showing that the null hypothesis for all these questions can be rejected and the mean precision of our method is significantly higher than the random baseline. The p-value for the first question shows that cohesion of the random features are not significantly different from those recommended by our method.

Results showed that in average, 91% of the recommended features were assessed as related to the overall goals of CoSS. While only 73% of the features were deemed to be cohesive, the reviewers classified 86% of them as useful, suggesting that some lack of cohesion did not undermine the utility of the

TABLE 9
Qualitative Analysis Results

Question	Null hypothesis	Average precision of the recommender	Average precision of the random baseline	p-value
Q_1	$\mu_{rand} = \mu_{rec}$	0.73030303	0.807575758	0.145440189
Q_2	$\mu_{rand} \geq \mu_{rec}$	0.674116162	0.17739899	5.43955E-06
Q_3	$\mu_{rand} \geq \mu_{rec}$	0.909343434	0.327020202	4.9452E-08
Q_4	$\mu_{rand} \geq \mu_{rec}$	0.768813131	0.327651515	3.90332E-07
Q_5	$\mu_{rand} \geq \mu_{rec}$	0.865782828	0.350505051	1.34156E-06

recommendations.

6 Threats to Validity

Threats to validity can be classified as construct, internal, external validity, and reliability. External validity evaluates the generalizability of the approach. The training data consists of product feature descriptors extracted from Softpedia. Our feature model therefore only includes features which are included in SoftPedia listings. However, as SoftPedia covers a wide range of product types, we assume the technique will generalize to other as yet unevaluated domains. Due to this limitation, the COSS case study was deliberately chosen to be compatible with domains covered by SoftPedia. If a non-compatible case study had been chosen it would have been difficult to generate useful recommendations. These limitations were clearly described in the text. In future work we intend to extend the training data to include product descriptions mined from additional sites such as GOOGLE Apps MarketPlace.

Construct validity evaluates the extent to which the construct that was intended to be measured was actually measured. In the evaluation of the IDC algorithm, we used four different answer sets. Each of these answer sets was created manually using card-sorting techniques, and represents one viable clustering of feature descriptors. It is unclear whether the various clustering algorithms would have performed equally well against all valid clusterings. Unfortunately the time and effort needed to manually create such clusterings made it infeasible to repeat the exercise multiple times. On the other hand, the clusterings were created by people otherwise entirely unrelated to our study, and their only instructions were to cluster feature descriptors into meaningful and distinct features. We did not tell them how many features to identify. Furthermore, the results from the IDC study were fairly consistent across all three of the studied datasets.

Another construct threat to validity is in our choice of leave-one-out evaluation for quantitatively evaluating the recommender system. While this is a standard technique for evaluating recommender systems, it suffers from the problem that it is only able to evaluate known information. For example, if a known feature is removed from a product, and the recommender system is able to successfully recommend it back in a certain ranking of the recommender features list, it counts as a successful recommendation at that ranking. On the other hand, our recommender system may recommend a feature which does not appear in the feature list of that product. This result will be recorded as an incorrect recommendation even though the recommended feature might be a viable recommendation for that product. This problem is particularly evident in our dataset, as the lists of features associated with each product are certainly incomplete. On the other hand, this problem has a negative impact on our results and so does not bias the validity of our approach. Despite this problem, our recommender system was able to recommend back a significant number of known features at high positions in the ranked list. Furthermore, we augmented our findings from this quantitative study with a qualitative case study in which generated recommendations were evaluated by human analysts.

Internal validity reflects the extent to which a study minimizes systematic error or bias, so that a causal conclusion can be drawn. We designed the COSS evaluation to reduce this bias. The evaluation was performed by members of our research team who had not otherwise been exposed to the datasets and the results of the recommendation system prior to this study. We performed a blind study in which evaluators (1) had not previously been exposed to the COSS recommendations made by our approach, and (2) which now included 50% of random recommendations. (3) The evaluators were not informed which features were made by our recommender system and which were random. (4) The evaluators were not informed of the number of random features in the recommendation set.

7 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 domain analysis methods such as FODA (Feature Oriented Domain Analysis) [2], FROM (Feature-Oriented Reuse Method) [3] , FAST (Family Oriented Abstraction, Specification, And Translation)[5], FeatureRSEB [33], and ODM (Organization Domain Modeling) [34] have been proposed to support the discovery, analysis, and documentation

of commonalities and variabilities within a domain. These methods rely on human intensive activities which are more appropriate for a small domain. Tools that have been constructed to support these methods [35], [36] [37][4] primarily focus upon modeling variability and managing it across different products. Therefore significant upfront manual effort is still required in order to perform a domain analysis, identify features, and create their composition rules. There has recently been considerable research on using data mining and machine learning techniques in order to construct domain models through organizing and discovering relationships between software requirements. For example, the Domain Analysis and Reuse Environment (DARE) [4] uses text processing tools to extract domain vocabulary from text sources, and then identifies common domain entities, functions, and objects, by clustering related words and phrases. These clusters are then used by the analyst to create a feature table. Chen et al. [38] manually constructed requirements relationship graphs (RRG) from several different requirements specifications and then used hierarchical clustering techniques to merge them into a single domain tree. However, this approach is not easily scalable beyond trivially sized domains because the domain user is required to manually create each requirements relationship graph. Although these graphs are constructed from one application and can be reused in other applications in the same domain, considerable user intervention is required to modify and adopt them. Vander Alves et.al. [39] utilized the Vector Space Model (VSM) and Latent Semantic Analysis (LSA) to determine the similarity between requirements and then generated an association matrix which is clustered using the Agglomerative Hierarchical Clustering (AHC) algorithm. Identified clusters are then merged together using a depth-first algorithm to compare nodes and create a feature model for the domain. Noppen et al. [40] extended this work by including fuzzy sets in the framework to allow individual requirements to be associated with multiple features. Niu and Easterbrook [41], [42] developed an on-demand clustering framework that provided semi-automatic support for analyzing functional requirements in a product line. They proposed an information retrieval (IR) and natural language processing (NLP) approach to identify important entities and functions in the requirements as a functional requirements profile (FRP). FRPs are identified according to the linguistic characterization of a domain and tend to capture the domain's action themes from the requirements documents. FRPs are used as clustering objects and information retrieval techniques are used to identify (model) the variability among FRPs. In this framework, human intervention is needed to validate FRPs and their attributes.

Several other researchers have used association rule mining to automate the construction of domain feature models from a set of already existing application features. Lora-Michiels [43] applied association rules analysis to identify mandatory and optional relationships; then performed a chi-square statistical test in conjunction with the association rules to identify *require* relationships. Similarly, She [44] used a set of individual product configurations consisting of a list of features, then applied conjunctive and disjunctive association rule mining to construct a probabilistic feature model. In other work, Acher et. Al [45] extracted feature models of several applications from tabular documentation of product features. They used a name based merging technique to merge the overlapping parts of each feature model, synthesize the variability information and build a single feature model representing the product family.

However, all of these approaches on feature configuration, assume that the list of features for each individual application is available for synthesis and constructing the final domain model.

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 [46], or collaborative data of other users' ratings [7], or knowledge rules of the domain [47], or hybrid approaches [48]. Substantial amounts of work have been done in the area of evaluating recommender systems [30] and on newer highly efficient algorithms such as those based on matrix factorization [49]. 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 [28], and a high-level overview of possible usages and applications of recommender systems in this domain [50]. Nevertheless, our use of recommender systems in this paper builds on the substantial background of research in this area.

8 Conclusion

This paper has presented a novel feature recommender system to support the domain analysis process. This is a critical early-phase part of the software development life-cycle and is essential in both application development and product line development [51], [52]. Our system mines feature descriptors for hundreds of products from publicly available software repositories of product descriptions and uses this data to discover features and their associations. For feature discovery, we proposed a novel Incremental Diffusive Clustering algorithm. The evaluation results presented in this paper, reveal that our clustering

method is more suitable for this task than some other well-known clustering methods. We expect that the results from our clustering analysis can be applied to other clustering problems in the requirements engineering domain. Another novel contribution 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. This has the advantage of augmenting an initially sparse product description before making a more extensive set of recommendations.

The performance of the recommender system was evaluated both through quantitative experiments and qualitative analysis. The results showed that our recommender system is capable of making viable feature recommendations to help domain analysts build domain models for a variety of software applications. In addition, the examples shown throughout the paper have illustrated the process, its benefits and its limitations.

Future work will focus on mining additional software repositories and directories to broaden the knowledge of the system, improve the interaction between the user and the system, explore techniques to enhance the navigation and visualization of the recommended features, and conduct more extensive qualitative evaluations with a wider group of project stakeholders.

ACKNOWLEDGMENTS

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

REFERENCES

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

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

[3] K. C. Kang, S. Kim, J. Lee, K. Kim, G. J. Kim, and E. Shin, "FORM: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, pp. 143–168, 1998.

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

[5] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," *Proc. of the 20th Intn'l Conf. on Very Large Data Bases*, 1994.

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

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

[8] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli, "On-demand feature recommendations derived from mining public software repositories," in *33rd International Conference on Software Engineering*. Honolulu, Hawaii, USA: ACM Computer Society, May 2011, p. 10.

[9] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge: Cambridge University Press, 2008.

[10] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. MA, USA: Kluwer Academic Publishers, 1981.

[11] I. S. Dhillon and D. S. Modha, "Concept decompositions for large sparse text data using clustering." *Machine Learning*, vol. 42, no. 1/2, pp. 143–175, 2001.

[12] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation." *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[13] 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. [Online]. Available: bib-papers/MDLN01a.pdf

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

[15] 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*, pp. 1073–1082, 2008.

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

[17] C. Duan, "Improving requirements clustering in an interactive and dynamic environment," DePaul University, Tech. Rep.

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

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

[20] K. M. Lewis and P. Hepburn, "Open card sorting and factor analysis: a usability case study," *The Electronic Library*, vol. 28, no. 3, pp. 401–416, 2010.

[21] M. Papagelis, D. Plexousakis, and T. Kutsuras, "Alleviating the sparsity problem of collaborative filtering using trust inferences." in *iTrust*, ser. Lecture Notes in Computer Science, P. Herrmann, V. Issarny, and S. Shiu, Eds., vol. 3477. Springer, 2005, pp. 224–239.

[22] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Analysis of recommendation algorithms for e-commerce," 2000, pp. 158–167, proc. ACM Conference on Electronic Commerce.

[23] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," *IEEE Computer*, vol. 40, no. 6, pp. 27–35, 2007.

[24] W. Lin, S. A. Alvarez, and C. Ruiz, "Efficient adaptive-support association rule mining for recommender systems," *Data Mining and Knowledge Discovery*, vol. 6, pp. 83–105, 2002. [Online]. Available: bib-papers/LAR02.pdf

[25] 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)*, Minneapolis, MN, October 2000, pp. 158–167. [Online]. Available: bib-papers/SKKR00a.pdf

[26] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data(SIGMOD'00)*, New York, NY, June 2000.

[27] 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*. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 37–46.

[28] 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*, pp. 1419–1426, 2009.

[29] E. Spertus, M. Sahami, and O. Buyukkokten, "Evaluating similarity measures: a large-scale study in the orkut social network." Chicago, Illinois, USA: ACM, 2005, pp. 678–684. [Online]. Available: http://portal.acm.org/citation.cfm?id=1081956

[30] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans. on Info. Sys.*, vol. 22, pp. 5–23, 2004.
[31] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," 2001, pp. 285–295.
[32] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-thieme, "L.s.: Bpr: Bayesian personalized ranking from implicit feedback," in *In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI*, 2009.
[33] M. Simos, D. Creps, C. Klingler, L. Levine, and D. Allemang, "Software technology for adaptable reliable systems (stars) organization domain modeling (odm) guidebook version 2.0," Manassas, VA: Lockheed Martin Tactical Defense Systems, Tech. Rep. STARS-VC-A025/001/00, 1996. [Online]. Available: http://citeseer.ist.psu.edu/700189.html
[34] J. Coplien, D. Hoffman, and D. Weiss, "Commonality and variability in software engineering," *IEEE Softw.*, vol. 15, no. 6, pp. 37–45, Nov. 1998. [Online]. Available: http://dx.doi.org/10.1109/52.730836
[35] R. Krut, "Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology," *Software Engineering Institute, Pittsburgh, Technical Report CMU/SEI-93-TR-11*, 1993.
[36] K. Czarnecki, M. Antkiewicz, Chang, S. Lau, and K. Pietroszek, "fmp and fmp2rsm: eclipse plug-ins for modeling features using model templates," in *Object-oriented programming, systems, languages, and applications (OOPSLA)*. San Diego, CA, USA: ACM, 2005, pp. 200–201.
[37] T. von der Massen and H. Lichter, "RequiLine: A Requirements Engineering Tool for Software Product Lines," in *International Workshop on Product Family Engineering (PFE)*. Siena, Italy: Springer Verlag, 2003.
[38] 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*, vol. 0, pp. 31–40, 2005.
[39] 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.*, pp. 67–76, 2008.
[40] J.Noppen, P. van den Broek, N. Weston, and A. Rashid, "Modelling imperfect product line requirements with fuzzy feature diagrams," *VaMoS*, pp. 93–102, 2009.
[41] N.Niu and S.Easterbrook, "Extracting and modeling product line functional requirements," *Proc. of the 16th Intn'l Requirements Eng. Conf.*, 2008.
[42] N.Niue and S.Easterbrook, "On-demand cluster analysis for product line functional requirements," *Proc. of the 12th Intn'l Software Product Line Conf.*, pp. 87–96, 2008.
[43] A. Lora-Michiels, C. Salinesi, and R. Mazo, "A method based on association rules to construct product line models." in *VaMoS*, ser. ICB-Research Report, D. Benavides, D. S. Batory, and P. Grnbacher, Eds., vol. 37. Universitt Duisburg-Essen, 2010, pp. 147–150. [Online]. Available: http://dblp.uni-trier.de/db/conf/vamos/vamos2010.html
[44] S. She, "Feature model mining," Master's thesis, University of Waterloo, Waterloo, 08/2008 2008. [Online]. Available: http://hdl.handle.net/10012/3915
[45] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions," in *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, ser. VaMoS '12. New York, NY, USA: ACM, 2012, pp. 45–54. [Online]. Available: http://doi.acm.org/10.1145/2110147.2110153
[46] M. Pazzani and D. Billsus, "Content-based recommendation systems," *The Adaptive Web*, pp. 325–341, 2007.
[47] R.Burke, "Knowledge-based recommender systems," *Encyclopedia of Library and Info. Sys.*, vol. 69, 2000.
[48] R.Burk, "Hybrid rec. systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, pp. 331–370, 2002.
[49] J. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, pp. 30–37, 2009.
[50] 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)*, pp. 32–39, 2009.
[51] S. Apel and C. Kästner, "An overview of feature-oriented software development," *Journal of Object Technology (JOT)*, vol. 8, no. 5, pp. 49–84, July/August 2009.
[52] A. Classen, P. Heymans, and P.-Y. Schobbens, "What's in a feature: A requirements engineering perspective," in *FASE*, 2008, pp. 16–30.