A Context-Aware Recommender System for Extended Software Product Line Configurations

[1]Juliana Alves Pereira, [1]Sandro Schulze, [1]Sebastian Krieter, [2]Márcio Ribeiro, [1]Gunter Saake
[1]University of Magdeburg, Germany; [2]Federal University of Alagoas, Brazil
{juliana.alves-pereira,sandro.schulze,sebastian.krieter,gunter.saake}@ovgu.de;marcio@ic.ufal.br

ABSTRACT

Mass customization of standardized products has become a trend to succeed in today's market environment. *Software Product Lines* (SPLs) address this trend by describing a family of software products that share a common set of features. However, choosing the appropriate set of features that matches a user's individual interests is hampered due to the overwhelming amount of possible SPL configurations. Recommender systems can address this challenge by filtering the number of configurations and suggesting a suitable set of features for the user's requirements. In this paper, we propose a context-aware recommender system for predicting feature selections in an extended SPL configuration scenario, *i.e.* taking non-functional properties of features into consideration. We present an empirical evaluation based on a large real-world dataset of configurations derived from industrial experience in the *Enterprise Resource Planning* domain. Our results indicate significant improvements in the predictive accuracy of our context-aware recommendation approach over a state-of-the-art binary-based approach.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; *Extra-functional properties*; *Software functional properties*;

KEYWORDS

Software Product Lines, Feature Model, Non-Functional Properties, Configuration, Recommender Systems

ACM Reference Format:
[1]Juliana Alves Pereira, [1]Sandro Schulze, [1]Sebastian Krieter, [2]Márcio Ribeiro, [1]Gunter Saake. 2018. A Context-Aware Recommender System for Extended Software Product Line Configurations. In *VAMOS 2018: 12th International Workshop on Variability Modelling of Software-Intensive Systems, February 7–9, 2018, Madrid, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3168365.3168373

1 INTRODUCTION

Software Product Lines (SPLs) are software platforms that can be personalized based on customers' requirements. A key part of an SPL is a model that represents features and their dependencies (*i.e.*,SPL configuration rules). This model can be extended by adding *Non-Functional Properties* (NFPs) as feature attributes resulting in *extended SPLs*. Deriving products from an extended SPL requires considering the configuration rules of the model and satisfying the product's functional and non-functional requirements. While numerous mature SPL configuration tools exist that ensure a consistent SPL configuration [20, 24], configuring large and complex SPLs with those tools is often beyond the users' capabilities of identifying valid combinations of features that match their (non-functional) requirements. Therefore, the configuration process may result in inappropriate or inefficient configurations. To overcome these limitations, we introduced the adoption of feature predictions in previous work [23]. Our previous approach relies on collaborative-based recommendation algorithms and binary data from previous configuration to identify relevant features. In this paper, we extend this idea by considering product requirements as additional contextual data and thereby improving the overall recommendation quality.

Our solution works interactively on a stream of (de)selections of features and uses *contextual modeling* to incorporates NFPs. This data has an essential advantage, that is, being adaptive to changes of user preferences and release of new features. Consequently, our system is able to capture currently relevant features for a user even though no configuration with these features have been observed in the past. The aim of our system is to reduce the users' configuration effort and enhance their configuration experience.

To summarize, we provide the following three contributions:

- We adopt a context-aware recommender system tailored for the extended SPL configuration scenario.
- We target a challenge in the recommender systems field, which is the recommendation of unexpected events (*e.g.*, new features).
- We conduct extensive experiments on a large real-world industrial SPL to evaluate the proposed approach.

For evaluating our approach, we formulate three research questions (*RQs*) to be answered by means of an experimental study:

- *RQ1.* Can a context-aware recommender system support the configuration process of extended SPLs in realistic scenarios?
- *RQ2.* Which are the recommendation quality benefits of a context-aware approach against a non-contextual approach?
- *RQ3.* What is the effect of using different combinations of contextual data?

To answer our RQs, we conduct numerous experiments with four context-aware recommendation algorithms on a real-world dataset of SPL configurations. To address *RQ1*, we compare the results from all algorithms with an interactive configuration process and a randomized approach. To address *RQ2*, we compare our context-aware recommender against a state-of-the-art recommender using binary data [23]. Finally, since our context-aware approach is intended to work in three main stages, *RQ3* follows analyzing the impact of each stage in the quality of recommendations.

VAMOS 2018, February 7–9, 2018, Madrid, Spain

ACM ISBN 978-1-4503-5398-4/18/02…$15.00
https://doi.org/10.1145/3168365.3168373

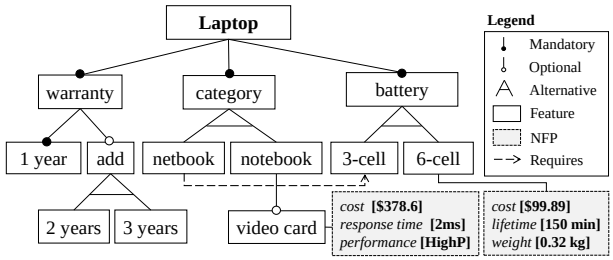


Figure 1: A sample of an EFM for a laptop product line.

The remaining paper is structured as follows: We present the relevant background in Sec. 2 and give an overview on related work in Sec. 3. We introduce our approach in Sec. 4 and describe evaluation design and results in Sec. 5 and 6, respectively. Finally, we conclude the paper and outline directions for future work in Sec. 7.

2 PRELIMINARIES

In this section, we formally introduce the terms *extended SPL*, *product configuration*, and *hybrid recommender* (RS); as well as a state-of-the-art recommender approach [23] extended in this paper.

Extended SPL. An SPL can be extended by adding NFPs as feature attributes, known as *extended SPL* [7]. An Extended SPL describes the dependencies and constraints among functional features and NFPs through an *Extended Feature Model* (EFM). An EFM is formally represented as a hierarchically arranged set of functional features composed by relationships organized as a tree-like structure. As a running example, Fig. 1 shows an EFM for a Laptop *Product Line* (PL) where a functional feature can be either mandatory (`1 year warranty`) or optional (`2` or `3 years warranty`), and an NFP can be either quantitative (`cost`) or qualitative (`performance`). Moreover, multiple NFPs can be associated to the same feature. In addition to the relations among features and NFPs, EFMs often contain *cross-tree constraints* (CTCs). CTCs add new feature interdependencies to the feature tree, by restricting the selection of non-directly connected optional features, *e.g.*, `netbook` $\rightarrow$ `3-cell`.

Definition 1. *An* EFM $\mathcal{EFM}(F, P, R)$ *is a tuple that consists of a set of n features* $F = \{f_1, f_2, ..., f_n\}$*, a set of m NFPs* $P = \{p_1, p_2, ..., p_m\}$*, and a set of k configuration rules* $R = \{\vec{r}_1, \vec{r}_2, ..., \vec{r}_k\}$*.*

A configuration rule $\vec{r}_i$ *represents a clause from the EFM's propositional formula in CNF, such that* $\vec{r}_i \in \{-1, 0, 1\}^n$ *and the component j of* $\vec{r}_i$ *specifies whether the feature j should be selected* ($r_{i_j} = 1$)*, deselected* ($r_{i_j} = 0$)*, or is not relevant* ($r_{i_j} = -1$) *within this constraint.*

Product Configuration. A product configuration refers to the process of selecting a set of optional features that comply with the configuration rules (*e.g.*, a laptop cannot have both `3-cell` and `6-cell battery`) and better fulfill the product's requirements.

Definition 2. *Given an* $\mathcal{EFM}$*, a* configuration $\vec{c} = (c_{f_1}, c_{f_2}, ..., c_{f_n})$ *represents a selection of features such that* $c_{f_i} = 1$*, if feature* f_i *is selected,* $c_{f_i} = 0$ *if it is deselected, and* $c_{f_i} = -1$ *if its state is undefined.*

Definition 3. *A configuration* $\vec{c}$ *is* complete *iff* $\vec{c}$ *defines each feature (i.e.* $\forall i \in \{1, .., n\} : c_{f_i} \neq -1$*), otherwise it is* partial*. For a given* $\mathcal{EFM}$*, we denote the set of all its complete configurations with* CC *and the set of all its partial configurations with* $\mathcal{PC}$*.*

Definition 4. *A configuration* $\vec{c}$ *is* valid *iff it satisfies all constraints in R when considering all undefined features in* $\vec{c}$ *as deselected (i.e.* $\forall \vec{r} \in R, \exists i \in \{1, .., n\} : r_i \neq -1$*), otherwise it is* invalid*. For a given* $\mathcal{EFM}$*, we denote the set of all its valid configurations with* $\mathcal{VC}$ *and the set of all its invalid configurations with* $\mathcal{IC}$*.*

At the end of the configuration process, the user's configuration must be complete and valid, as well as satisfying all product's requirements. Product's requirements are calculated by aggregating the NFPs of all selected features by considering feature interactions. Interactions occur when combinations among features share a common component or require additional component(s), affecting NFP values [29]. As an example, configurations that include both features `6-cell` and `video card` may have a global negative impact in the NFP `lifetime`. The unexpected result is caused by the additional battery consumption of `video card`.

In a preliminary survey [19], we identified two types of product's requirements: *resource constraints* and *users' preferences*. *Resource constraints* are decision rules with regards to product's limitations, such as budget=$550.00. They complement the interdependencies expressed through the EFM, restricting the set of valid configurations. In addition, *users' preferences* allow the user's specification of NFPs relative importance, such as minimize `cost` and maximize `performance`. Thus, among all valid product configurations $\mathcal{VC}$, stakeholders desire the optimal one that can meet multiple resource constraints and users' preferences.

A Feature-Based Recommender for SPL Configuration. To support the SPL configuration task, recommender systems can provide suggestions that effectively prune the large configuration space so that users are directed towards features that best meet their needs and preferences. In previous work, we proposed the use of a personalized feature-based recommender that relies on configurations from previous users to generate personalized recommendations for a current user [23]. In particular, we adapted *Collaborative Filtering* (CF) algorithms to predict whether a user will like a particular feature. The CF-based system uses a *configuration matrix* $X_{U \times F}$ as input, where U represents the set of users whose preferences are known for the set of features in F. Therefore, given a set of n complete and valid previous configurations $\{\vec{c}_1, ..., \vec{c}_n\} \subseteq CC \cap \mathcal{VC}$, the matrix $X_{U \times F}$ is defined as:

$$X_{U \times F} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,m} \end{bmatrix}$$

Our previous approach aims at predicting the relevance of a set of undefined features $f \subseteq F$ for a current user with a partial configuration $\vec{pc} \subseteq \mathcal{PC}$, based on likely relevant features from X. Notice that $\vec{pc}$ is continuously augmented as the user interacts with the system over time guiding him on how to get a complete configuration.

Though our previous approach has shown to be useful for supporting the SPL configuration process, it exhibits a limitation: It only examines binary variables, indicating the selection of a feature in previous (user) configuration(s). Hence, it is sensitive to the number of features configured for a user in the past (known as *sparsity problem* [1]). This problem can be resolved by combining multiple techniques within the recommender system [8].

Hybrid Recommender. To overcome the sparsity problem faced by the previous approach, we combine four recommendation techniques: *(i) context-aware, (ii) knowledge-based, (iii) CF-based*, and

(iv) rule-based. The *context-aware recommender* attempts to suggest features based on inferences about the user's needs and preferences. It has contextual knowledge about product's requirements, *e.g.*, the financial context of a user. In addition, the *knowledge-based recommender* builds a complete utility function from historical data. The utility function infers many different factors that contribute to the value of a configuration by weighting the significance of each feature for each user, such as the popularity of new features by computing the similarity with other features and analyzing historical data, rather than just selected features. Then, the *CF-based recommender* recognizes similarities between users on the basis of previous configurations and generates recommendations. Finally, the *rule-based recommender* recommends only features that satisfy the configurations rules from an EFM.

To summarize, we use knowledge data from features, users, and configurations to assign weights to the matrix X based on contextual data from product's requirements. Next, we use this as input to a CF-based recommender that will recommend features that satisfy the configurations rules from an EFM and better meet the product's requirements. Hence, contextual-aware and knowledge-based systems don't suffer from the sparsity problem because they do not (only) rely on binary data about previous configurations.

3 RELATED WORK

Recommender systems reduce the complexity of comprehension tasks and help to get insights for making decisions [26]. Recommendation techniques have been studied by the SPL research community to support several tasks beyond configuration [13, 17]. In this paper, we focus on recommendations to guide the configuration process and we acknowledge several works in this field.

Several approaches address the configuration in extended SPLs by using *dynamic decision models* [5, 11, 15, 30, 32]. Through decision models, the user can interactively construct a complete preference function by weighing the significance of each relevant feature in terms of satisfying their non-functional requirements. However, as real-world EFMs tend to be inherently large and complex, this often creates a significant burden of interaction, *i.e.*, the user has to assess several types of variability relations among features and NFPs. Furthermore, as product's requirements may be conflicting and one feature may contribute to many requirements, users may still be unsure about their preferences.

To overcome the limitations of the above approaches, several authors have proposed optimization techniques to automatically support the configuration process [3, 6, 12, 14, 22]. However, these techniques usually return a set of optimal configurations and none of them guides the user in selecting the most appropriate one. Hence, these techniques can be complementary to our approach that aims at guiding the user in the selection of a single configuration.

In previous work, we proposed the single use of binary data from previous configurations to generate personalized recommendations [23]. However, this approach prevents the recommendation of new facts or new perspectives that would be valuable to the user. Thus, to improve the quality of the recommendations, we propose in this paper to incorporate multiple recommendation techniques so that it allows the specification of additional contextual data in the form of product's requirements. Similar to related works [19], our current approach is mainly built on the idea that feature priorities may change, based on the target stakeholders and the context of the configuration.

4 CONTEXT-AWARE RECOMMENDER

To overcome the information overload, generated by state-of-the-art SPL configuration tools, we propose a hybrid recommender system. It uses the users' explicit requirements, previous configuarations, and implicit data of features and users to predict the relevance of unselected features in a given configuration. Our approach allows the user to freely specify contextual information in the system. Then, the system attempts to model and learn users' preferences automatically by interactively obtaining preference feedback on their partial configuration. In Fig. 2, we present a general overview of the configuration process, which consists of five main activities: (1) *specify contextual data*, (2) *configure product*, (3) *check product completness*, (4) *compute predictions*, and (5) *prediction adjustments*.

(1) First, the user *explicitly* specifies contextual data by collecting the product's requirements from stakeholders. As we focus on the recommendation process, we assume that the users are capable of understanding the goals of the target system and translate those goals into product's requirements. For example, laptops for gamers differ from laptops targeting other types of customer profiles (*i.e.*, low performance laptops would not meet the needs of a gamer due to the high processing demand of current games). We are aware that one of the key problems of product configuration is to find out what actually is the stakeholders' requirements [25]. This, however, goes beyond the scope of this paper.
(2) Next, a pre-filtering stage builds a filter to include only historical data pertaining to the user-specified criteria in which the recommendation is relevant. Then, the user selects features of interest from a configuration view on the list of ranked features from our recommender system. Each time the user configures a particular feature, decision propagation strategies are automatically applied to validate the configuration (see [21] for more details) and the ranked list of relevant features is updated.
(3) Subsequently, we check the completeness of the configuration by verifying whether there are some undefined features. In case we have a partial configuration, we compute feature predictions.
(4) To compute predictions, the prediction modeling stage uses knowledge-based information directly in the recommendation algorithm (Sec. 4.2) as an explicit predictor of a user's preference.
(5) Then, the post-filtering stage reorders the recommended features by weighting the predictions with the probability of NFPs relevance in the user's specific context.

4.1 Contextual Modeling

In our approach, contextual data can be obtained: *explicitly*, *implicitly*, and by *inferring*.

- Explicitly: a set of previous configurations, a current partial configuration, resource contraints, and user's preferences.
- Implicitly: domain expert judgement to specify NFPs [5, 16, 18].
- Inferring: use of functional metrics to a static or runtime quantifiable measurement of NFPs [9, 27, 29, 31], which allows the detection of interactions resulting from a valid set of features.

We assume that *implicitly* and *inferring* NFPs were previously specified by using state-of-the-art techniques. For instance, Cruz et al. [9] and Zanardini et al. [31] infer NFPs based on a rigorous

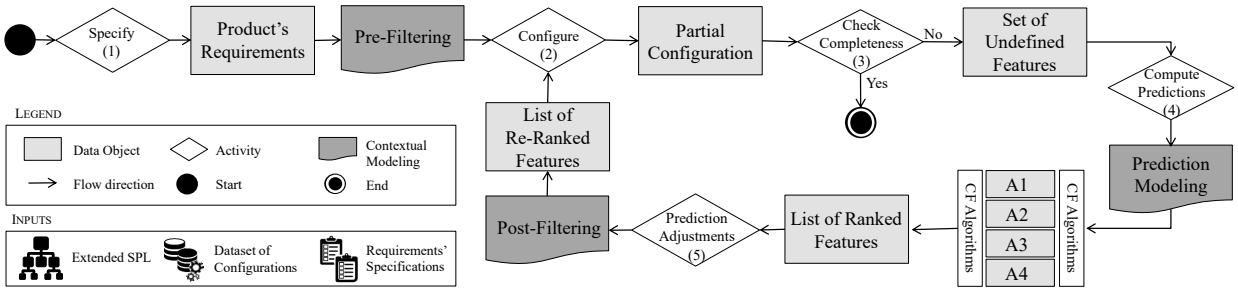


Figure 2: An overview of our feature-based recommender approach.

static source code inspection, *e.g.*, analysis of coupling and cyclomatic complexity. In this case, the success of implicitly specifying or inferring NFPs depends very significantly on the quality of such techniques, and it also varies considerably across different systems.

Pre-Filtering. This stage takes as input a user's specifications of *resource constraints* as pre-filtering data. To this end, the user specifies resource constraints based on implicitly and inferring NFPs from features and configurations. The aim of this stage is to filter out noisy or irrelevant data before they are used for computing recommendations [1]. An example of contextual pre-filter data for the laptop PL in Fig. 1 would be: ensure that the `cost` and `response time` of the product do not exceed $550.00 and 4 ms, respectively.

This pre-filter data is then used to reduce the initial matrix X (Sec. 2), containing data about previous configurations, to a matrix W using the following equation:

$$X_{U\times F} \rightarrow W_{U\times F}^{D[cost\leq 550, time\leq 4](User, Feature, Config)} \quad (1)$$

where $[cost \leq 550, time \leq 4]$ denotes two contextual pre-filters, and $D[cost \leq 550, time \leq 4](User, Feature, Config)$ denotes the historical configuration dataset obtained from D by keeping only the set of data where *cost* and *time* constraints are satisfied.

By using this reduction-based approach, we may not have enough data for accurate features' prediction. This is the case because this stage builds a local prediction model for a particular context, and this may limit the power of the predictions due to few remaining data [1]. To overcome such limitation, we use the n-dimensional Euclidean metric [2] to search for similar configurations to the ones in the matrix W based on data from configurations' NFPs. Then, to have a significant amount of historical data, *i.e.* filters with more than N configurations, we add a set of the highest similar configurations to the matrix W, where N is some predetermined threshold (*e.g.*, 10% of the dataset was used in that study).

Prediction Modeling. After filtering the set of relevant configurations and features, contextual data is used directly inside the recommendation-generating algorithms as part of the features' prediction estimation. Unlike the traditional recommender systems that deal with ratings, our knowledge-based recommender relies on the notion of weights as a way to measure the *utility* of a feature for a user. Thus, we assign a utility weight w for all selected features in W. The general formula to measure the *utility* w of feature j for user i is given as:

$$w(i,j) = \frac{Freq(i,j)}{Period(i,j)} \quad (2)$$

where $Freq$ and $Period$ are information inferred by the system. $Freq$ represents how often user i used feature j and $Period$ the number of days since the last usage of feature j by user i, which measures the popularity of feature j. Following the idea that features have a short life cycle, this measurement is important to prevent the recommendation of outdated features. Moreover, features' popularity drift over time. Thus, the system is capable to capture such signals and timely adapt its recommendations accordingly. However, on this scenario, the recommendation of unexpected events, *e.g.* new features, are no longer possible. To overcome this limitation, we calculate an additional utility weight s for all deselected features in W, based on their *similarity* with other features' NFPs.

Since the range of values among NFPs may vary widely (*e.g.*, 1-60,000 ms for `response time` and $50-1550 for `cost`), we firstly use a feature scaling method to *normalize* the range of NFP values to a common scale in the range of $[0, 1]$. Then, we calculate the *weighted arithmetic mean* s for deselected features in W:

$$s(i,j) = \sum_{k=1}^{m} c_{i,k} \cdot \frac{Sim(f_j, f_k)}{\sum_{l=1}^{m} Sim(f_j, f_l)} \cdot \frac{1}{m} \quad (3)$$

where m represents the number of features, and $Sim(f_j, f_k)$ measures the *similarity* between a target feature j and another feature k based on their normalized NFP values. In our experiments, we use the n-dimensional Euclidean metric [2] to calculate similarity. Once the weights w and s are estimated for the matrix W, traditional recommendation algorithms are used to compute the list of the k highest features' predictions for an active user (Sec. 4.2).

To summarize, we follow four main steps: *(i)* assign a utility weight w for all selected features; *(ii) normalize* features' NFPs; *(iii)* calculate the *similarity* between features based on their normalized NFPs; and *(iv)* calculate a *weighted arithmetic mean* s for deselected features based on their similarity weight with the other features. This stage makes the recommender system to continuously adapt to the set of new features and to discard outdated features. Therefore, even without any historical knowledge about the features' utility, recommendations can be done by modeling and inferring a weight s for historical data. Thus, the use of contextual data in the prediction modeling stage avoids problems of users with a few set of selected features (known as *cold-start problem* in recommender systems [1]).

Post-Filtering. In this stage, contextual information is used after computing recommendations to adjust the resulting set of predictions, *i.e.,* when generating the final ranked list of relevant features. We use a model-based post-filtering approach [1], in which we build a predictive model by weighting the predicted features with an additional probability of relevance based on the contextual data from *users' preferences.* According to Asadi et al. [4], users' preferences are categorized into six levels: high, medium, and low *positive*; and

low, medium, and high *negative*. In case the user classifies an NFP p_i as a *positive influence* over the computed predictions, we weight the computed predictions by considering the values 1 (high), 0.66 (medium), and 0.33 (low) as an additional probability of relevance a that is *directly proportional* to the predicted value. In case the user classify an NFP p_j as a *negative influence*, we consider the values 1 (high), 0.66 (medium), and 0.33 (low) as an additional probability of relevance b that is *inversely proportional* to the predicted value. To summarize, the relevance score $Rel(\vec{pc}, f)$ computed in Sec. 4.2 is updated using the equation:

$$Rel(\vec{pc}, f) = \frac{Rel(\vec{pc}, f) \cdot \sum_{i=1}^{n} a_i \cdot p_{i_f}}{\sum_{j=1}^{m} b_j \cdot p_{j_f}} \tag{4}$$

where n and m correspond to the number of NFPs with positive and negative influences, respectively. As an example, for the laptop PL in Fig. 1, the user may mention that a high `performance` laptop has a *high positive* effect over the product's requirements, while a high `response time` has a *high negative* effect. On the *optimization objective* scenario, the user wish to minimize the system's `response time`, while maximizing the system `performance`. Consequently, a feature with high performance and fast response time is more preferable and relevant than another feature with slightly higher prediction but lower performance and higher response time.

4.2 Collaborative-Based Recommender

We adapt four traditional CF-based recommender algorithms to the extended SPL configuration scenario: (A1) *User-Based CF*, (A2) *Feature-Based CF*, (A3) *User-Based Average Similarity*, and (A4) *Feature-Based Average Similarity*. Given the matrix W and a new partial configuration $\overrightarrow{pc} \in \mathcal{PC}$ that is currently being configured by a target user, these algorithms are used to estimate unknown features' preferences for $\overrightarrow{pc}$. The first two algorithms are neighborhood-based CF approaches, which require the definition of a distance function between users or features, respectively [10]. We make a comparative analysis of the two main types of neighborhood-based algorithms: *user-based* and *feature-based CF*. In addition, we conduct experiments with the *user-based* and *feature-based average similarity* (AS) algorithms to investigate if restricting the neighbourhood size has an influence on the quality of recommendations in our application.

A1. User-Based CF. The main idea of a user-based CF algorithm is that similar users have similar patterns of configurations, therefore similar features receive similar weights. First, we search for similar users (*i.e.*, nearest neighbours) to a target user in the matrix W. A similar user is a previous user, who has weights attributed to features (a row vector from $W_{U \times F}$) that are similar to the ones from a target user according to a similarity measure. To find the most similar users, we use either the *Pearson Correlation Coefficient* (PCC) similarity measure or the cosine similarity. For a detailed explanation of both measures we refer to [2]. A set of feature weights from a previous user qualifies as similar, if $Sim(\overrightarrow{pc}, \overrightarrow{c_i}) > \tau$, where τ is a similarity threshold that is given as an input parameter (Sec 5.2). If a similarity measure exceeds a given threshold τ, then the corresponding configuration is considered a neighbour. All neighbours build a *neighbourhood* $\mathcal{N}(\overrightarrow{pc}, \tau) = \overrightarrow{c_i} \in CC \wedge sim(\overrightarrow{pc}, \overrightarrow{c_i}) > \tau$.

Second, once the neighborhood $\mathcal{N}(\overrightarrow{pc}, \tau)$ has been determined, the algorithm calculates the relevance score $Rel(\overrightarrow{pc}, f)$ of an undefined feature f for a target configuration $\overrightarrow{pc}$ as the weighted average of her neighborhood weights for the feature f. The overall prediction function is as follows:

$$Rel(\overrightarrow{pc}, f) = \bar{pc} + \frac{\sum_{\overrightarrow{c_i} \in \mathcal{N}(\overrightarrow{pc}, \tau)} Sim(\overrightarrow{pc}, \overrightarrow{c_i}) \cdot (c_{i,f} - \bar{c_i})}{\sum_{\overrightarrow{c_i} \in \mathcal{N}(\overrightarrow{pc}, \tau)} |Sim(\overrightarrow{pc}, \overrightarrow{c_i})|} \tag{5}$$

where $\overrightarrow{pc}$ represents the set of features which an active user has selected and $\bar{c_i} = (\sum_{j \in \overrightarrow{pc}} c_{i,j})/(|\overrightarrow{pc}|)$ is the mean weight (analogously for $\bar{pc}$) for each user i for the specified set of j selected features in $\overrightarrow{pc}$. Note that the sum in the formula iterates over the neighbours of $\overrightarrow{pc}$, where each prediction is weighted with the similarity weight of its owner to the target user.

A2. Feature-Based CF. In feature-based CF, the neighbourhood is constructed in terms of features (or columns in the weighting matrix W) rather than users. The weighted average value of these (raw) utilities is reported as the predicted value.

In particular, we calculate the weight similarity between a target feature f for which the predictions are being computed and all the other features. Similarly, we use PCC and cosine similarity measurements and denote the *neighbourhood* of a target feature f as $\mathcal{N}(\overrightarrow{f}, \tau)$. Then, the weighted average of the neighbours weightings is used to compute the prediction of feature f for the target user $\overrightarrow{pc}$. The relevance score $Rel(\overrightarrow{pc}, f)$ is calculated as follows:

$$Rel(\overrightarrow{pc}, f) = \frac{\sum_{\overrightarrow{f_i} \in \mathcal{N}(\overrightarrow{f}, \tau)} Sim(\overrightarrow{f_i}, \overrightarrow{f}) \cdot pc_{f_i}}{\sum_{\overrightarrow{f_i} \in \mathcal{N}(\overrightarrow{f}, \tau)} |Sim(\overrightarrow{f_i}, \overrightarrow{f})|} \tag{6}$$

The basic idea of this algorithm is to leverage the user's own weights on similar features when making the prediction.

A3. User-Based AS. This algorithm uses the same principle as CF, but it does not use the notion of a neighbourhood. Consequently, all weights attributed to features in the matrix W are considered for computing the relevance score of a feature for a target user. Accordingly, Eq. 5 is changed to:

$$Rel(\overrightarrow{pc}, f) = \bar{pc} + \frac{\sum_{\overrightarrow{c_i} \in W} Sim(\overrightarrow{pc}, \overrightarrow{c_i}) \cdot (c_{i,f} - \bar{c_i})}{\sum_{\overrightarrow{c_i} \in W} |Sim(\overrightarrow{pc}, \overrightarrow{c_i})|} \tag{7}$$

Note that the sum iterates over all the users in W. This means that the relevance score of feature f is an average similarity of all other users' weights attributed to feature f.

A4. Feature-Based AS. Similar to the user-based AS recommender, Eq. 6 is changed to:

$$Rel(\overrightarrow{pc}, f) = \frac{\sum_{\overrightarrow{f_i} \in W} Sim(\overrightarrow{f_i}, \overrightarrow{f}) \cdot pc_{f_i}}{\sum_{\overrightarrow{f_i} \in W} |Sim(\overrightarrow{f_i}, \overrightarrow{f})|} \tag{8}$$

Here the sum iterates over all features in W. This means that the relevance score of feature f is an average similarity over all other features' weights attributed to the same target user $\overrightarrow{pc}$.

In the last step, the relevance scores $Rel(\overrightarrow{pc}, f)$ are returned to the post-filtering stage. In this stage, to provide an optimized guidance for the user, features are ranked with respect to user's preferences.

5 EXPERIMENT DESIGN

This section describes the experiment design to evaluate the four recommendation algorithms introduced in Section 4.2.

5.1 Target Software Product Line and Dataset

We evaluate the effectiveness of our configuration approach by applying it to a real-world dataset of 2,000 configurations and 203 features from our business partner in the ERP domain. It delivers an application scenario where customers, features, and configurations are described as relations having the following attributes:

- Customer: receives the feature recommendations; defined as Customer(CustomerID, Name, Location, Market Domain, Type).
- Feature: set of all the features that can be recommended; defined as Feature(FeatureID, Cost, Profit, Provider, Category).
- Configuration: set of selected features for a previous customer; defined as Configuration(CustomerID, FeatureID, Frequency of Usage, Date Last Usage).

The contextual information consists of the following specifications:

- Resource constraints: market domain, location, and type.
- Stakeholders' preferences: minimize the system's cost for the customer, while maximizing the system profit for the company.

To evaluate our algorithms, on this dataset, we performed an offline evaluation that encompasses three main steps: *(i) parameter optimization*, *(ii) splitting into training and test datasets*, and *(iii) evaluation metrics*. Next, we discuss each one of this steps.

5.2 Parameter Optimization

The implemented algorithms require the specification of a similarity measure and a threshold value τ to define which users are considered neighbours. To find the optimal parameters to make the algorithms fit an unknown dataset, we perform an initial optimization step. This step is essential as it may influence the quality of the recommendations considerably. For example, since we are considering using either PCC or cosine similarity as the similarity measure to predict features' relevance, we must compare the performances of these two methods to determine which of them produce the best predictive model. Therefore, we hold out a random sample of 50% of all configurations from our original dataset and run a genetic algorithm. The genetic algorithm was used to optimize the F-Measure of the four used recommendation algorithms (Sec. 5.4). Every parameter in this phase was validated using a *10-fold cross validation*. The optimal similarity measure for all algorithms are *cosine similarity* and the approximately optimal threshold values τ are 0.7576 and 0.5661 for both, user-based CF and feature-based CF algorithms respectively. These parameter settings are used in our main evaluation and applied to the remaining 50% of configurations.

5.3 Splitting into Training and Test Datasets

In this main evaluation phase, we use the *leave-one-out evaluation* protocol. According to this protocol one configuration is used for testing and the remaining ones are given to the algorithm as training data (*i.e.* all other configurations). Formally, the specified entries of the configuration matrix $\overrightarrow{pc} \notin X$ are referred to as the training data, whereas the partial configuration $\overrightarrow{pc} \in \mathcal{PC}$ is referred to as the test data. It simulates the behavior of an active user configuring a single target product in a configuration system, where the remaining configurations are available to the system to assist him in finding the features matching their individual preferences and expectations. However, to further simulate the interactive configuration process by an active user, the set of features is randomly partitioned into 10 equal sized sub-sets. Then, we increasingly give a sub-set of (de)selected features from a test configuration to the recommender system as training data and the remaining ones are hidden from the algorithm and used as testing, *e.g.*, 10% of a complete configuration is used as training and 90% as test data. To perform well, the system has to recommend the features that were hidden from the algorithm.

To ensure reliability, the cross-validation process is repeated 1,000 times (*i.e.*, where 1,000 represents the remaining 50% of the configurations from our original dataset) for each algorithm with each configuration used exactly once as the testing data. To produce a single estimation, the final quality measure of a recommendation algorithm is then the average quality over all configurations.

5.4 Evaluation Metrics

Once the recommender system returned a ranked list of relevant features, we perform a quality measurement of the recommendations. Firstly, we compare the real set of truly relevant features *Rel* known from the test configuration with the set of recommended ones *Rec* using *precision* and *recall* as quality measure. Since we approach recommendation as a ranking task, we are mainly interested in relatively few most relevant features. Thus, *precision* and *recall* are computed based on the top-10 ranked features (*i.e.*, $w = 10$) which is common in recommender systems [28]. While *precision* measures the proportion of recommended features that were truly selected, *recall* measures the proportion of all truly selected features that appear in the top-w ranked features. They are calculated as follows: $Precision = \frac{|Rec \cap Rel|}{w}$ and $Recall = \frac{|Rec \cap Rel|}{|Rel|}$. Consequently, the optimal value for both measures is 1.0, indicating that all truly selected features have been correctly recommended, without any deselected features among the recommendation. Since our goal is to maximize both, precision and recall, in our evaluation we use a measure that combines both, *i.e.*, the *F-Measure*:

$$F\text{-}Measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{9}$$

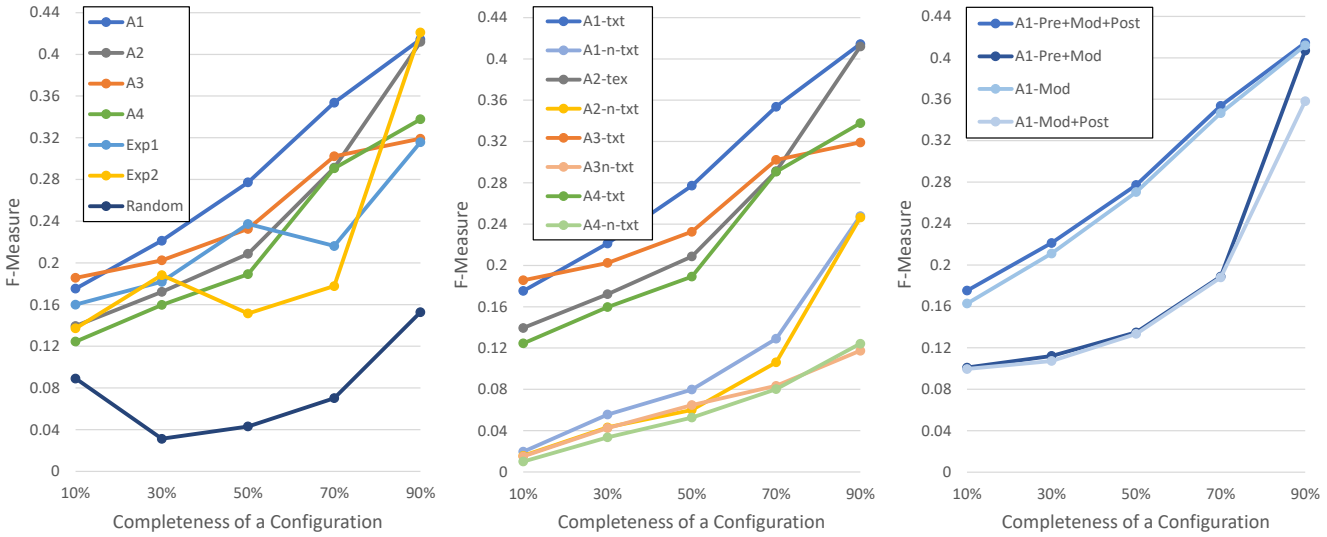
The F-Measure is high, when both precision and recall are high. Thus, the quality of a recommendation is good, if the top-w predicted features overlaps with the set of truly selected features from the test configuration.

6 RESULTS AND DISCUSSION

In this section, we investigate how efficient and effective the proposed recommender system is in supporting the SPL configuration process by answering the research questions *RQ1–RQ3* introduced in Sec. 1. The experiments reported in this section intend to present a preliminary observation on the feasibility of our approach.

6.1 Approach Effectiveness

We evaluate the effectiveness of the algorithms by comparing them with a random baseline method that simulates the performance of an uninformed user without any support from a recommender system. This method recommends randomly chosen non-selected features and, therefore, it indicates the minimal performance level every algorithm should reach. However, as in most real-world applications no human is fully uninformed, to additionally indicate how useful our recommendation algorithms are, we reported also the results from an interactive configuration process from two domain



a) Approach effectiveness. b) Context-aware approach benefits. c) Different combinations of contextual data.

Figure 3: F-Measure achieved by different methods.

experts (*Exp1* and *Exp2*). In this context, for each testing configuration, we informed the expert the target user context, as well the training set of pre-selected features. Then, for each percentage of features that were given to them as training data, they choose the features that most suited the specified requirements.

Fig. 3a presents the *F-Measure* achieved by the seven methods: a *random* baseline method, four *recommendation algorithms*, and two *domain experts*. F-Measure combines recall and precision, *i.e.* higher values are better. On the horizontal axis of the figure, we present the completeness of a configuration, *i.e.* the percentage of features that were given to the method as training data, where only the remaining part of the configuration needed to be predicted.

We observe that all methods presented good results outperforming the baseline random recommender. For all algorithms, we observe an increase of performance as the configuration becomes more complete. This is because the algorithms receive more data for training or rasoning. The user-based CF algorithm achieves the best performance over all the other algorithms at nearly all stages of the configuration process, except for the initial part of a configuration, when few information about the current configuration is available. Overall, CF algorithms yielded a better performance tham AS algorithms. One potencial reason is that AS algorithms do not utilize any neighborhood information. In addition, in average the CF algorithms provided even better results tham the interactive configuration performed by domain experts. Moreover, domain experts are engaged in a time-consuming and tedious task.

6.2 Context-Aware Approach Benefits

In this section, we estimate how effective our approach is in comparison with the previous non-contextual version of our approach [23]. Therefore, we run the non-contextual version of the four implemented algorithms in our target dataset. It is worst mentioning that we do not use the datasets from [23], since these datasets do not work with NFPs. In Fig. 3b, we report the *F-Measure* achieved by both approaches for each of the four algorithms.

The contextual recommendation algorithms significantly outperforms the non-contextual algorithms at all the stages of the configuration process. The main reason is the benefit of having additional relevant data for calculating unknown features' relevance, instead of only having binary information. The non-contextual version of the algorithms are limited by the users that are explicitly associated with the features that they recommend and therefore has difficulty when the space of selected features is sparse (*i.e.*, in average few users have selected the same features). Sparsity is a significant problem in the SPL configuration domain, since there are many features available and, unless the configuration dataset is very large, the chances that another user will share a large number of selected features is small. Consequently, pure CF-based recommenders work best on datasets where the density of user configuration is relatively high across a small and static number of features. However, which of these two trends dominates depends on the application domain and on the specifics of the available data. Therefore, we plan at extending a state-of-the-art configuration tool through a 5-star feature classification by using the contextual data only for those contextual situations where this method outperforms the standard non-contextual version of the same algorithm. Thus, the approach proposed in this paper is expected to perform equally well or better than the state-of-the-art approach [23] in practice.

6.3 Different Combinations of Contextual Data

In Sec. 4.1 we described the main contextual stages under which our approach is developed. Since not all stages might be useful for recommendation purposes, in this section we empirically evaluate the effect that each stage has in the quality of the recommendations.

Since the effect was proportionally the same for all stages, in Fig. 3c we show the results for the most effective algorithm (*i.e.*, user-based CF in Sec. 6.1). We observe that all proposed algorithms achieved the best performance when making use of all available contextual data. It is evident that explicitly modeling a large amount of data significantly boosts the recommendation performance under the same algorithm. Furthermore, note that although the A1-Mod

recommender is outperformed by the A1-Pre+Mod+Post recommender, the results from both methods are quite similar. Therefore, this implies that inappropriate contextual modeling in the pre- and port-filtering stages can even hurt the performance.

We would also like to point out that an accurate configuration predictions unquestionably depends on the degree of which the recommender system incorporates the relevant contextual information. There are several approaches, *e.g.* from machine learning, data mining, and statistics, to determine the relevance of a given type of contextual information. These approaches aim at screening all the NFPs and filtering out those that do not affect a particular recommendation application. This, however, goes beyond the scope of this paper and remains as an important next step, which is part of our future work.

7 CONCLUSION AND FUTURE WORK

In this paper, we propose a hybrid recommender approach. Our approach adopts traditional CF recommendation algorithms to estimate features' preferences based on users' contextual information. The proposed approach is not only able to support new users' configurations, but also provides configuration upgrades for previous users. To assess its effectiveness, we conduct experiments in a large real-world dataset of SPL configurations. We empirically show that our context-aware approach significantly outperforms the corresponding non-contextual approach in terms of quality of recommendations.

As future work, we intend to explore various types of statistical tests to identify which of the contextual NFPs are truly significant in the sense that they indeed affect the recommendations. Also, we plan to evaluate our approach under the use of other recommendation algorithms on self-adaptive systems. By testing our approach on other systems, we can investigate how the diversity of application scenarios and the number of previous configurations affect the quality of the recommendations. Moreover, since recommender algorithms are frequently intended to work on very large datasets of configurations, we also aim at analyzing the impact of the proposed algorithms on configuration performance. Finally, we aim at conducting a user-controlled study of our approach to investigate the user's satisfaction with the recommendations.

ACKNOWLEDGMENT

This work was partially funded by the CNPq grants 202368/2014-9 and 307190/2015-3, CAPES/PROCAD 175956, CAPES/PGCI 117875, and FAPEAL PPGs 60030 000435/2017.

REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin. 2011. Context-aware recommender systems. In *Recommender systems handbook*. Springer, 217–253.
[2] Xavier Amatriain and Josep M Pujol. 2015. Data mining methods for recommender systems. In *Recommender Systems Handbook*. Springer, 227–262.
[3] Mohsen Asadi, Ebrahim Bagheri, Dragan Gašević, Marek Hatala, and Bardia Mohabbati. 2011. Goal-driven software product line engineering. In *SAC*. ACM, 691–698.
[4] Mohsen Asadi, Samaneh Soltani, Dragan Gasevic, Marek Hatala, and Ebrahim Bagheri. 2014. Toward automated feature model configuration with optimizing non-functional requirements. *IST* 56, 9 (2014), 1144–1165.
[5] Ebrahim Bagheri and Faezeh Ensan. 2014. Dynamic decision models for staged software product line configuration. *Req. Engineering* 19, 2 (2014), 187–212.
[6] David Benavides, Pablo Trinidad Martín-Arroyo, and Antonio Ruiz Cortés. 2005. Automated reasoning on feature models.. In *CAiSE*, Vol. 5. Springer, 491–503.
[7] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: a literature review. *Information Systems* 35, 6 (2010), 615–708.
[8] Robin Burke. 2002. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction* 12, 4 (2002), 331–370.
[9] Jonathas Cruz, Pedro Santos Neto, Ricardo Britto, Ricardo Rabelo, Werney Ayala, Thiago Soares, and Mauricio Mota. 2013. Toward a hybrid approach to generate software product line portfolios. In *2013 IEEE Congress on Evolutionary Computation*. IEEE, 2229–2236.
[10] Christian Desrosiers and George Karypis. 2011. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender Systems Handbook*. Springer, 107–144.
[11] José A Galindo, Deepak Dhungana, Rick Rabiser, David Benavides, Goetz Botterweck, and Paul Grünbacher. 2015. Supporting distributed product configuration by integrating heterogeneous variability modeling approaches. *Information and Software Technology* 62 (2015), 78–100.
[12] Jianmei Guo, Jia Hui Liang, Kai Shi, Dingyu Yang, Jingsong Zhang, Krzysztof Czarnecki, Vijay Ganesh, and Huiqun Yu. 2017. SMTIBEA: A hybrid multi-objective optimization algorithm for configuring large constrained software product lines. *Software & Systems Modeling* (2017), 1–20.
[13] Mostafa Hamza and Robert J Walker. 2015. Recommending features and feature relationships from requirements documents for software product lines. In *RAISE*. IEEE Press, 25–31.
[14] Robert M Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. 2016. SIP: Optimal product selection from feature models using many-objective evolutionary optimization. *TOSEM* 25, 2 (2016), 17.
[15] Marcello La Rosa, Wil MP van der Aalst, Marlon Dumas, and Arthur HM Ter Hofstede. 2009. Questionnaire-based variability modeling for system configuration. *Software and Systems Modeling* 8, 2 (2009), 251–274.
[16] Lucas Machado, Juliana Pereira, Lucas Garcia, and Eduardo Figueiredo. 2014. Splconfig: Product configuration in software product line. In *CBSoft*. 1–8.
[17] Ana C Marcén, Jaime Font, Óscar Pastor, and Carlos Cetina. 2017. Towards feature location in models through a learning to rank approach. In *SPLC*. ACM, 57–64.
[18] Lina Ochoa, Oscar González-Rojas, and Thomas Thüm. 2015. Using decision rules for solving conflicts in extended feature models. In *SLE*. ACM, 149–160.
[19] Lina Ochoa, Juliana Alves Pereira, Oscar González-Rojas, Harold Castro, and Gunter Saake. 2017. A survey on scalability and performance concerns in extended product lines configuration. In *VaMoS*. ACM, 5–12.
[20] Juliana Alves Pereira, Kattiana Constantino, Eduardo Figueiredo, and Gunter Saake. 2016. Quantitative and Qualitative Empirical Analysis of Three Feature Modeling Tools. In *ENASE*. Springer, 66–88.
[21] Juliana Alves Pereira, Sebastian Krieter, Jens Meinicke, Reimar Schröter, Gunter Saake, and Thomas Leich. 2016. FeatureIDE: scalable product configuration of variable systems. In *ICSR*. Springer, 397–401.
[22] Juliana Alves Pereira, Lucas Maciel, Thiago F Noronha, and Eduardo Figueiredo. 2017. Heuristic and exact algorithms for product configuration in software product lines. *ITOR* 24, 6 (2017), 1285–1306.
[23] Juliana Alves Pereira, Pawel Matuszyk, Sebastian Krieter, Myra Spiliopoulou, and Gunter Saake. 2016. A feature-based personalized recommender system for product-line configuration. In *GPCE*. ACM, 120–131.
[24] Juliana Alves et al. Pereira. 2015. A systematic literature review of software product line management tools. In *ICSR*. Springer, 73–89.
[25] Rick Rabiser, Paul Grünbacher, and Martin Lehofer. 2012. A qualitative study on user guidance capabilities in product configuration tools. In *ASE*. ACM, 110–119.
[26] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. *Introduction to recommender systems handbook*. Springer.
[27] Luis Emiliano Sánchez, J Andrés Diaz-Pace, Alejandro Zunino, Sabine Moisan, and Jean-Paul Rigault. 2014. An approach for managing quality attributes at runtime using feature models. In *SBCARS*. IEEE, 11–20.
[28] Guy Shani and Asela Gunawardana. 2011. Evaluating recommendation systems. In *Recommender Systems Handbook*. Springer.
[29] Norbert Siegmund, Sergiy S Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting performance via automated feature-interaction detection. In *ICSE*. IEEE Press, 167–177.
[30] Lei Tan, Yuqing Lin, and Li Liu. 2014. Quality ranking of features in software product line engineering. In *APSEC*, Vol. 2. IEEE, 57–62.
[31] Damiano Zanardini, Elvira Albert, and Karina Villela. 2016. Resource–usage–aware configuration in software product lines. *Journal of Logical and Algebraic Methods in Programming* 85, 1 (2016), 173–199.
[32] Guoheng Zhang, Huilin Ye, and Yuqing Lin. 2014. Quality attribute modeling and quality aware product configuration in software product lines. *Software Quality Journal* 22, 3 (2014), 365–401.