International Journal of Computational Intelligence System
Vol. 14(1), 2021, pp. 1880–1894
DOI: https://doi.org/10.2991/ijcis.d.210620.003; ISSN: 1875-6891; eISSN: 1875-6883
https://www.atlantis-press.com/journals/ijcis/

Research Article

Predictive Analytics for Product Configurations in Software Product Lines

Uzma Afzal[1], Tariq Mahmood[2], Raihan ur Rasool[3], Ayaz H. Khan[4,*], Rehan Ullah Khan[5], Ali Mustafa Qamar[6]

[1]*Computer Science Department, Federal Urdu University of Arts, Science and Technology, Karachi, Pakistan*
[2]*Computer Science Department, Institute of Business Administration, Karachi, Pakistan*
[3]*Centre for Applied Informatics, Institute for Sustainable Industries & Liveable Cities, Engineering and Science, Victoria University, Melbourne, Australia*
[4]*Computer Science Department, Habib University, Karachi, Pakistan*
[5]*Department of Information Technology, College of Computer, Qassim University, Buraydah, Saudi Arabia*
[6]*Department of Computer Science, College of Computer, Qassim University, Buraydah, Saudi Arabia*

ARTICLE INFO

Article History
Received 11 Jan 2021
Accepted 27 May 2021

Keywords
Software product line,
Predictive analytics,
Data science,
Feature model,
Inconsistency,
Information system

ABSTRACT

A Software Product Line (SPL) is a collection of software for configuring software products in which sets of features are configured by different teams of product developers. This process often leads to inconsistencies (or dissatisfaction of constraints) in the resulting product configurations, whose resolution consumes considerable business resources. In this paper, we aim to solve this problem by learning, or mathematically modeling, all previous patterns of feature selection by SPL developers, and then use these patterns to predict inconsistent configuration patterns at runtime. We propose and implement an informative Predictive Analytics tool called predictive Software Product LIne Tool (*p*-SPLIT) which provides runtime decision support to SPL developers in three ways: 1) by identifying configurations of feature selections (patterns) that lead to inconsistent product configurations, 2) by identifying feature selection patterns that lead to consistent product configurations, and 3) by predicting feature inconsistencies in the product that is currently being configured (at runtime). *p*-SPLIT provides the first application of Predictive Analytics for the SPL feature modeling domain at the application engineering level. With different experiments in representative SPL settings, we obtained 85% predictive accuracy for *p*-SPLIT and a 98% Area Under the Curve (AUC) score. We also obtained subjective feedback from the practitioners who validate the usability of *p*-SPLIT in providing runtime decision support to SPL developers. Our results prove that *p*-SPLIT technology is a potential addition for the global SPL product configuration community, and we further validate this by comparing *p*-SPLIT's characteristics with state-of-the-art SPL development solutions.

1. INTRODUCTION

Software Product Lines (SPLs) [1–3] are used to configure software products in which different sets of features are configured and then integrated by different teams of product developers. This is called "product configuration" wherein each team selects its feature sets from a domain-engineered Feature Model (FM) having one or more constraints [4–6]. Product configuration itself happens during the application engineering phase. When individual feature sets are integrated to form the final product, inconsistencies can occur because domain FM constraints can collectively remain unresolved due to selection of contradictory features. Much effort is then required by teams to reconfigure their feature sets, which is a complicated, resource-intensive, and a well-known problem in SPL industry [5,7–12]. A need to solve this problem during application engineering has always been highlighted by major SPL industries [13–23].

Some Artificial Intelligence (AI) [24] techniques have since been used to solve this problem, albeit during domain engineering to ensure that the FM itself remains consistent with respect to the features being added by developers. For instance, in [25], the authors have translated the FM into description logic to resolve internal inconsistencies, and in [26,27], the authors have done the same using abductive reasoning and knowledge-base (KB) rules, respectively. A review of these AI techniques (from 1990 to 2009) is given in [28]. However, we have been unable to discover any AI-based solution to resolve inconsistencies during product configuration at runtime (using a consistent FM) at the application engineering level. Rather, the focus of SPL industry in this case has been mostly on identifying inconsistencies [8]. This research gap becomes more critical as inconsistencies can also arise due to a lack of communication between developers, a change of requirements during the configuration, and the iterative transfer of inconsistencies from one development stage to another [5].

In this paper, we focus on a novel idea that the process of selection of features for any SPL product has a characteristic regularity or

[*] *Corresponding author. Email: ayazhk@gmail.com; ayaz.hassan@sse.habib.edu.pk*

pattern, similarly to how different patterns of human behavior can be easily detected in online shopping and other human-centric applications. Our idea is to create an information system that can detect and memorize, rather *learn*, these patterns through applicable and well-known mathematical models. In this way, the selected model can be trained to have complete knowledge of how features have been selected up to now, specifically, the different steps of feature selection that have been followed by all individual SPL developers. In addition, such a system can then identify which types of patterns lead to inconsistent or consistent configurations, and provide this information to SPL developers who are currently configuring products at runtime. In fact, AI has seen a rapid pace of research since the publication of [28], and it is possible to implement our idea by using the Predictive Analytics (PA) technology [29–31], commonly known as data science or machine learning, to predict inconsistencies which can occur in the future with a given probability at runtime [10] (detailed in Section 2.2). Our contribution is the development of an information system for SPL product configuration, called *p*-SPLIT (*p*redictive Software Product LIne Tool), which uses PA to extract patterns or regularities from historical product configuration data for both consistent or inconsistent configurations. It then uses these patterns to make runtime predictions about upcoming inconsistencies during future product configurations. This is the first application of PA to application engineering phase of SPL product configuration to resolve the inconsistency-related problems. In *p*-SPLIT, we employ the Random Forest (RF) algorithm [32] to classify and distinguish between inconsistent and consistent patterns of feature selection, because it is efficient, robust to noise and outliers, its output patterns are actually visible and comprehensible, and has shown to outsmart at least 170 other PA algorithms in a comprehensive evaluation [33] and in other papers [32,34–36].[1]

From aforementioned discussion, we derive the following single research question which we address thoroughly in this paper: **RQ: What is the possible value which PA technology can bring to the SPL configuration process to resolve the issue of occurrence of inconsistencies at runtime during application engineering?** To answer RQ, we implemented and tested *p*-SPLIT on real-world, historical, product configuration data obtained from a representative, anonymous multinational SPL organization (more details in Section 3). We also conducted a subjective evaluation of *p*-SPLIT's results with industry-based SPL designers, who validated the usefulness of our tool with respect to usability, effectiveness, efficiency, and user satisfaction. We show that *p*-SPLIT is able to extract patterns of inconsistent and consistent product configurations from the historical data. These patterns can be generalized to future product configurations and are capable of indicating those runtime configurations which can potentially become inconsistent (or consistent) later on. The designers' subjective responses to these patterns do indicate the latter's potential for implementation in SPL industry.

Our anonymized dataset sample and the programming code are available at sites.google.com/site/afzaluzmaa/research/i-split/p-split which provides the more critical functions of *p*-SPLIT to allow reproduction of our experiments. The rest of the paper is organized as follows. In Section 2, we present the relevant background and in Section 3, we provide State-of-the-Art. The proposed *p*-SPLIT tool is detailed in Section 4, along with the experimental methodology. In Section 5, we present the results and evaluation of *p*-SPLIT. Finally, we conclude and present the future work in Section 6.

[1]This has been proven in several of our industrial projects conducted in the following companies: https://nexdegree.com/, https://codexnow.com/, and https://frontiertechnologyinstitute.com/

2. RELEVANT BACKGROUND

In this section, we first describe the configuration of a SPL and then present different concepts of PA.

2.1. Configuring the SPL Product

During the application engineering phase, an application developer configures an SPL product by selecting a set of features from the domain-engineered FM, according to the user preferences and predefined constraints (e.g., the selection of a single feature from an alternative group of features) [37]. In this paper, we use the following notation to describe a product configuration [8]:

- γ denotes an SPL. An SPL product configured from γ is denoted by Ψ; the nth product configured from Ψ is denoted by Ψ_n.
- F denotes a feature; a γ and/or Ψ can have a set of n features $\{F_1, F_2, \cdots, F_n\}$.
- The curly braces denote a product configuration, for instance, if a product Ψ_1 has features $F_1, F_2, \cdots, F_n$ then $\Psi_1 = \{F_1, F_2, \cdots, F_n\}$. Similarly, the parentheses denote the children features, for instance, if F_1 and F_2 are children of F_3, then $\Psi_1 = \{F_3(F_1, F_2)\}$.

We will now use an exemplary FM presented in [7] to describe the SPL product configuration. Specifically, the FM shown in Figure 1 represents the Vendor Master (VM) module of an Enterprise Resource Planning (ERP) SPL. An ERP is a large-scale information system which integrates different business units into a single information system. Each business units automates a specific department, like human resource management, product sales and purchase, production, warehousing, customer master, and VM. VM automates the business process related to the vendors that supply a business enterprise. For more details on ERP, please refer to [38]. The description of the FM in Figure 1 is as follows:

- F_1 is a mandatory feature, and F_6, F_8, and F_{11} are optional features.
- F_1 conveys *Vendor Information* in account, and has two children F_2 and F_3.
- F_2 and F_3 are "Anded" and represent *Vendor Name* and *Vendor ID*, respectively.
- F_3 has two children F_4 and F_5; F_4 generates *Manual* ID; and F_5 generates *System-Generated* ID.
- F_4 and F_5 exclude each other as *System-Generated* and *Manual* IDs cannot coexist in a valid product.
- F_6 is an optional feature to inactivate a vendor after a specific time period, i.e., F_7.
- F_6 includes F_7 to be a complete meaningful feature.

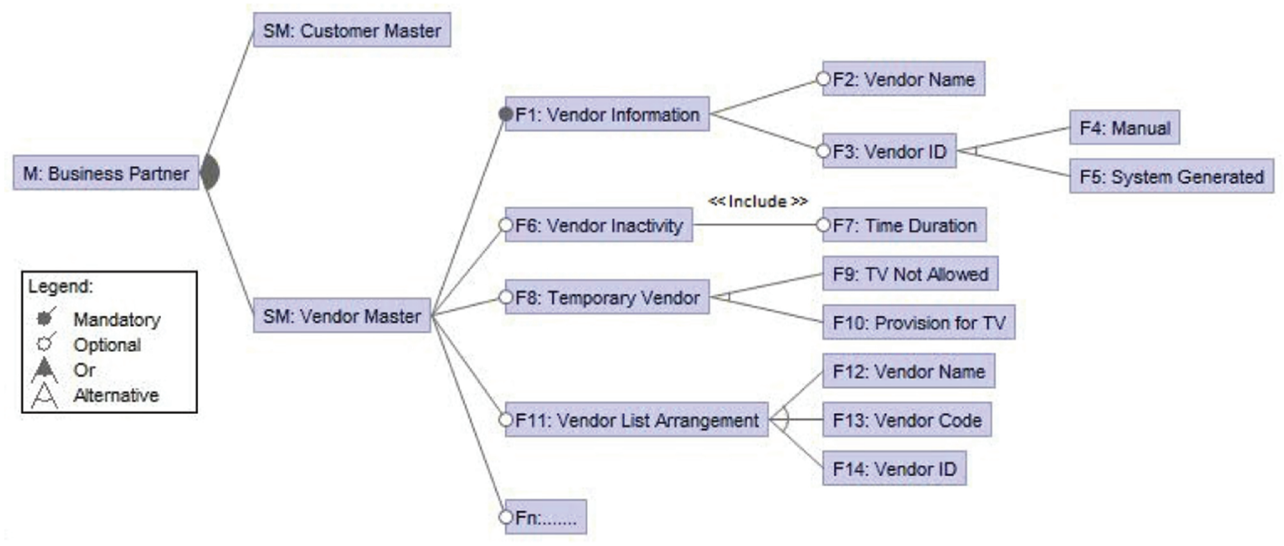


Figure 1 The vendor master feature model (adapted from [7]).

- F_8 takes the *Temporary Vendor* feature in account and has two children F_9 and F_{10}.
- F_9 and F_{10} exclude each other; F_9 allows temporary vendor while F_{10} does not allow it.
- F_{11} arranges the Vendor List and has three alternative children: F_{12}, F_{13}, and F_{14}.
- F_{12}, F_{13}, and F_{14} arrange vendors by their *Name*, *Code,* and *Postal Code*, respectively.

Using this FM of VM module as a reference, following are the examples of some consistent SPL product configurations (compliant with the constraints):

- $\Psi_1 = \{F_1(F_2, F_3(F_4)), F_6(F_7), F_8(F_9), F_{11}(F_{13})\}$
- $\Psi_2 = \{F_1(F_2, F_3(F_5)), F_8(F_{10}), F_{11}(F_{12})\}$

An SPL product configuration becomes inconsistent, if the selected features violate the predefined constraints (not compliant with FM). Again using FM of VM as a reference, following are some inconsistent SPL products:

- $\Psi_3 = \{F_6(F_7), F_8(F_9)\}$ is an inconsistent SPL configuration because the mandatory feature F_1 is missing.
- $\Psi_4 = \{F_1(F_2, F_3(F_5)), F_8(F_9, F_{10})\}$ is also an inconsistent configuration because features F_9 and F_{10} cannot coexist in the same configuration.

2.2. Predictive Analytics

As we already defined that PA is an advanced data analytics technology used to make predictions about unknown future events. It integrates knowledge from stochastic processes, mathematical modeling, machine learning, information technology, and business management, and has a diverse application domain including customer relationship management, clinical decision support system, direct marketing, customer retention, risk analysis, fraud detection, and recommender system [29–31,39–41]. In our work, we target PA's *classification* process.

Classification maps input data to output predictions based on a model inferred from a given dataset. Assume an input dataset with n rows (examples) with each example recording data over m features (or attributes). One of the features $C \in m$ is termed as the *label (Class)* feature and the set $P \in m$ of remaining features is termed as *predictors*. Classification is defined as the mathematical mapping $P \to C$. Ideally, a good classification algorithm like RF is able to autonomously select the subset of predictors $p \in P$, which can be used to predict C. The classification process is divided into two phases. In the *training phase*, we train potentially different models on subsets of input data and in the *testing phase*, we test and validate these models on other (unseen) subsets of the data to evaluate each predictor. By analyzing and selecting useful predictors, we fit the best-performing model to the available data based on performance measures like accuracy and recall (described later on in this section).

2.2.1. RF algorithm

There are many machine learning algorithms available to implement PA. These algorithms use different statistical approaches and hence their effectiveness varies in different scenarios [32,39,40]. As described in Section 1, we have selected the RF algorithm which constructs a multitude of Decision Trees (DTs) as predictive models [32]. DTs use concepts of entropy and information gain to select the features which are most useful in distinguishing between the labels (values) of the class variable C. The idea is to start from the complete classification dataset and keep on adding features until we get an accurate predictive model in the form of a tree. In this context, the entropy $E(D)$ for a classification dataset D is defined as $E(D) = \sum_{x \in X} p_x log_2 p_x$, i.e., the amount of randomness or unpredictability of D, where X is the total number of examples in D and $p(x)$ represents the probability of occurrence of a particular example x. The information gain $IG(D, A)$ represents the change in entropy of the dataset D when we consider feature F (for classification): $IG(D,F) = E(D) - E(D,F)$ where $E(D,F) = \sum_{j=1}^{f} \frac{|D_f|}{|D|} * E(D)$ where f represents the total number of possible values of F, $|D|$ represents the total number of examples in D, and $|D_f|$ represents the number of rows containing the particular value f of F.

Algorithm 1: Pseudocode for Generating a Decision Tree for the Binary Classification

```
Data: Input a classification dataset D with class
      variable C and set of predictors P
do
    if all examples have C = 'yes' then
        return leaf node as 'yes'
    end
    if all examples have C = 'no' then
        return leaf node as 'no'
    end
    Calculate entropy $H(D)$
    for each predictor x in P do
        Calculate Entropy $H(D,x)$ and $IG(D,x)$
    end
    Select predictor with maximum value of
    $IG(D,x)$
    Remove this predictor from P
while No predictor is left in P or the decision tree
  has all leaf nodes;
```

In a DT structure, the leaves represent the class labels (values of C) and the branches represent conjunctions of features that lead to the prediction of these class labels. A branch is hence made up of a combination of internal nodes, each representing a feature from the set of predictors. The number of branches emanating from each internal node represent the set of values of the feature at that node. A DT is constructed in a recursive, top-down manner from the training dataset D in which each row is labeled with a class label. In order to select a feature at each internal node, a typical DT employs Algorithm 1. Specifically, if all examples of D belong entirely to any one of the class labels, then that label is returned on the leaf node (as the prediction) and the DT terminates. Otherwise, the entropy and information gain of each feature is calculated and the feature with maximum information gain is selected as the root node. Then, the selected attribute is removed from D and the whole process repeats recursively on this modified dataset to generate the next feature on each branch of the current node.

As an example, consider the classification dataset shown in Figure 2 in which "Play Golf" is the binary class variable and the other features are weather-related predictors. The variable "Day" is an incremental counter and cannot be considered as a predictor which can have an influence on the decision to play golf. The classification problem is to use these 14 daily weather recording data to learn a DT through which we can predict whether we should play golf for a weather occurrence that will happen in the future.

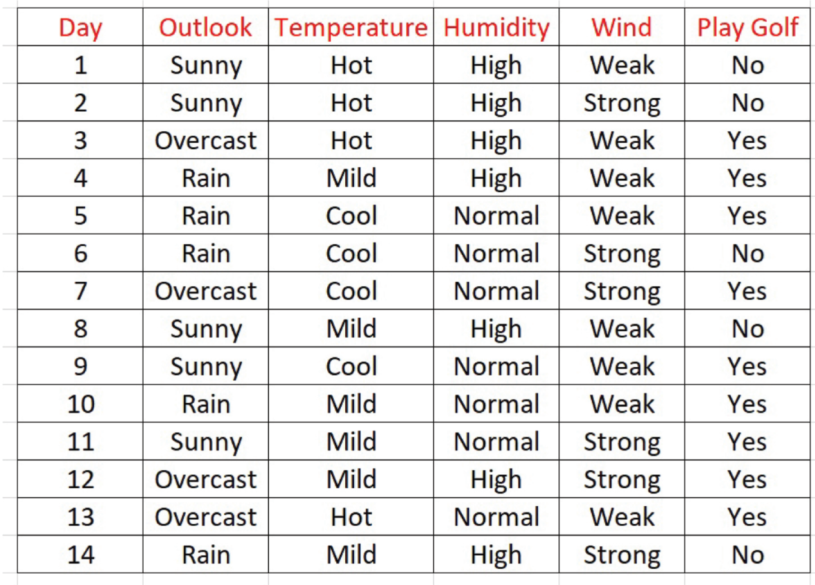
The entropy of D is 0.940 using $E(D) = \sum_{x \in X} p_x log_2 p_x$ with 9 and 5 examples for C = "Yes" and C = "No," respectively. As an entropy of 1 means an equal distribution of class labels, we conclude that the distribution of "Yes" and "No" in D is fairly random. We now estimate the information gained by adding each of the predictor features, starting with $IG(D, Wind)$ where $Wind$ has two values "Weak" and 'Strong." We calculate $H(Wind = Weak) = 0.811$ (8 examples have a 'Weak" value out of which 6 are for "Yes" and 2 for "No") and $H(Wind = Strong) = 1.000$ (6 examples have a "Weak" value, out of which 3 are for "Yes" and 3 for "No"). Now, $IG(D, Wind) = 0.940 - (8/14) * 0.811 - (6/14) * 1.000 = 0.048$. Similarly, we calculate $IG(D, Outlook) = 0.246$, $IG(D, Temperature) = 0.029$, and $IG(D, Humidity) = 0.151$. Hence, $Outlook$ with the highest information gain is selected at the root node. This process repeats recursively with the final DT shown in Figure 3. Therefore, on some given day, if the outlook is sunny and humidity is normal, then the prediction is to play golf and if it is raining with strong winds, then the prediction is not to play golf; however, if the winds are not that strong, then golf can still be played.

Two well-known methods to classify a typical DT are boosting and bagging [42,43]. In boosting method, an extra weight is given to those examples which have been incorrectly predicted by the earlier predictors. In bagging methods, successive trees are constructed using a bootstrap sample which are independent of the earlier trees. RF provides an additional layer of randomness to the bagging method. A typical DT splits each node using the best split among all variables; while RF splits a node using the best among a subset of variables randomly selected for that particular node, which makes it robust against overfitting. RF is simpler in nature because it uses only a few parameters, such as the number of DTs in the RF and the number of variables in the random sample at each node. Algorithm 2 presents the pseudocode of the RF which starts from the bootstrap sampling of the dataset, followed by the selection of the best split from a random sampling of predictors. Finally, it grows DT from the split.

Algorithm 2: Random Forest—Pseudocode [34]

```
Generate n bootstrap samples
for each n in sample do
    Randomly sample m of the predictors
    Choose the best split
    Grow the Tree
end
Aggregate the prediction of trees
Predict new data from aggregation
```

As discussed earlier, RF does not grow a single tree. Rather, it generates a collection of DTs which help in the visual and explicit representation of decisions and decision making. Similar to a typical DT, DTs in RF also construct models that can predict the value of a target variable based on several input variables. Here, each interior node corresponds to one of the input variables and the edges connect child nodes, so that all the possible values of that input variable are represented. Each leaf represents a value of the target variable on the basis of the values of input variables, represented by the path from the root to the leaf. Traversing different branches of the DT from the root node to the leaves provides different attribute combinations which classify the class labels.



Day	Outlook	Temperature	Humidity	Wind	Play Golf
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Figure 2 Classification dataset for playing golf.

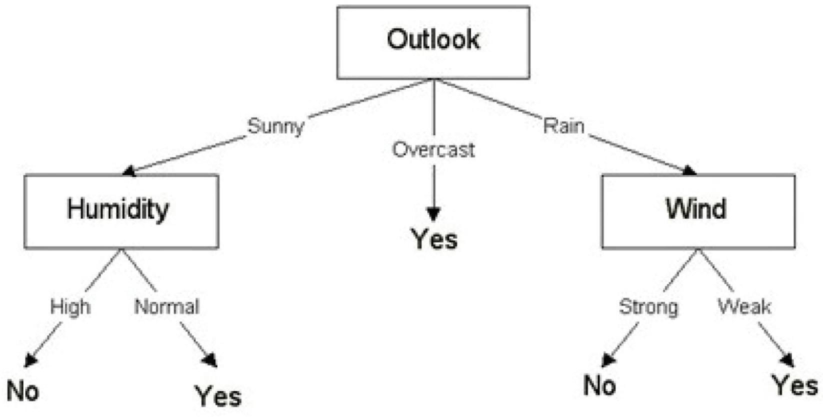


Figure 3 Final decision tree for predicting the playing of golf.

2.2.2. RF evaluation

We evaluated RF on the basis of six representative performance measures, namely Accuracy, Classification Error (CE), Kappa, Area Under the Curve (AUC), Precision, and Recall. Accuracy measures the systematic errors and can be calculated as:

$$\frac{\text{Number of Correct Predictions}}{|(TS)_D|} \tag{1}$$

where $(TS)_D$ is the total number of predictions in testing dataset. Also, CE measures the number of predictions incorrectly classified and can be calculated as:

$$\frac{\text{Number of Incorrect Predictions}}{|(TS)_D|} \tag{2}$$

Precision, recall, kappa and AUC are also good classifier measures for SPL product inconsistency problem owing to its binary nature. The decision made by the binary classifier can be represented in a structure known as confusion matrix, which contains four categories, i.e., True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). TP and TN represent the number of correct classification of positive and negative examples, respectively. Similarly, FP and FN represent the number of incorrect classification of positive and negative examples, respectively. Precision and recall, which measure the exactness and completeness of the classifiers, respectively, are then defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4}$$

Because of the slightly imbalanced nature of our datasets (most of the data belong to one class), we also used Kappa statistics and AUC to measure the classification performance [40]. Kappa is calculated as:

$$\frac{Ko - K_e}{1 - K_e} \tag{5}$$

Table 1 | SPL Issuesol; column "Ref." is the citation number; "Problem Solved" means the SPL problem solved.

Ref.	Problem Solved	Technology Used to Solve the Problem
[44]	Variability Mining	Recommendation System
[45]	Requirements Analysis	Framework Based on Cluster Analysis
[46]	Product Configuration	Feature Subset Recommendation based on Association Rule Mining
[47]	Manufacturing	Neural Network and Rules Induction
[48]	Product Requirements	Decision Model
[49]	Feature Model Design	Text Mining
[6]	Product Variability	Semantic Analysis
[25]	Product Inconsistencies	Description Logic; tested on limited feature set
[27]	Product Inconsistencies	Abductive Reasoning; Only identifies the inconsistencies, tested on limited feature set
[26]	Product Inconsistencies	Knowledge-base; focuses on inconsistent feature model
p-SPLIT	Product Inconsistencies	Predictive Analytics; detailed in next Section

where

$$K_e = \frac{(TN + FN) \times (TN + FP) + (FP + TP) \times (FN + TP)}{(TS)_D^2} \tag{6}$$

and

$$K_o = \frac{TN + TP}{(TS)_D} \tag{7}$$

Finally, AUC plots the TP rate vs the FP rate as the threshold value for item classification is 0 or is increased from 0 to 1. The TP rate increases quickly for the good classifier, and for the bad one, it increases linearly.

3. SPL: STATE-OF-THE-ART

Table 1 presents the state-of-the-art techniques to cater the SPL issues including inconsistent product configuration. In Noorian *et al.* [25], the authors present a description logic-based framework to manage the inconsistencies by identifying and resolving them. The framework is tested using a limited feature set, i.e., 35 features. An inconsistent configuration is passed to the framework which identifies and fixes the inconsistencies and generates a minimal set of consistent features. High identification and resolution times are recorded for large-scale feature set. Trinidad and Cortés [27] propose an abductive reasoning approach to identify an inconsistency with the possible reason. The proposed solution does not fix inconsistencies, moreover, an exemplary FM with a limited set of features is used for validation. Elfaki *et al.* [26] present a knowledge-based solution to fix the inconsistencies. The primary objective of the research is to correct an inconsistent FM due to dead and inconsistent features. For this, the given FM is converted into a KB to generate a list of inconsistent and dead features. An exemplary FM with 35 features is used to test the solution.

From a general perspective, there are several research papers which strongly motivate the application of novel IT and CS-related technologies to solve software-related problems for the customers. For instance, the work done in [50] stressed the importance of using AI in order to model the platform development process in SPLs. In our case, platform development is synonymous with feature modeling, and AI is synonymous with PA, which can be easily considered a sub-branch of AI in the context of Machine Learning [24]. Besides this, the importance of using data mining (a traditional name for PA) to predict the customers' business requirements in advance was stressed in [51,52]. In these papers, the authors propose a customer relationship framework or model that uses data mining to anticipate in advance the business requirements of the customer. In our case, we are doing the same by using PA to predict SPL features, which are synonymous with requirements.

Also, the work done in [53] has critically evaluated the impact of web-based PA (data mining) tools for the software industries and the clients. It mentions that such a venture can face complex issues, e.g., risk of investment, reduced budget, difficulty of communicating between different stake holders, and reduced knowledge of data mining outputs and processes.

From a technical perspective, the research articles applying PA or related technologies to SPL-related issues are quite limited (Table 1). Perhaps the work most related to our approach is the one by Kastner *et al.* [44], in which the authors introduce a new approach called *variability mining*. Here, given the domain knowledge of SPL features and the corresponding programming code, an internal representation of feature mappings and code structures is first built. Based on this model, a recommendation system is then used to recommend (or mine) the fragments of code, which the SLP developer should consider for configuring the product. Developers have the independence to accept or reject the recommendations, along with incorporating domain knowledge manually in the process. The system obtains a high average recall of 90% but a low average precision of 42%.

In another related paper [45], the authors present a framework based on cluster analysis [30] to analyze functional requirements in the SPL development process. This outputs different clusters of feature selections, particularly based on the perspectives of the stakeholders.

Another important work is presented in [46], where the authors use association rule mining to recommend the potential subset of features to configure an SPL product at runtime. The authors validate their work in an industrial setting but do not implement any comprehensive decision support system that uses the results in a more usable and personalized manner. The authors have not presented the results in enough detail to clearly understand the impact of the proposed technology in an industrial setting. Also, the work done in [47] validates the use of data mining techniques to solve complex problems in semiconductor wafer manufacturing SPLs. Some major issues include nonlinear interactions between different design groups, fast-changing business processes, a large variety of products, and the increasing volume of data (big data). The authors demonstrate how self-organizing neural networks and rule induction [24,30] can be used to solve these problems and increase the yield from 3% to 15% along with solving problems 10 times more efficiently.

In a somewhat more historical work [48], the author proposes the design and implementation of a specification infrastructure for SPLs, which can be reused by SPL developers later on to configure products in a better way. The specification can be structured to suit the needs of the customers, including a systematic method of reuse. The authors present a limited case study to validate the technology. However, it cannot handle the complex variability of SPL design, which is definitely not a hindrance for our PA-based approach, given that PA can also be based on techniques from Big Data Analytics and provides more robust models as compared to a simple specification.

In Maazoun *et al.* [49], the authors explore text mining technique to design the FM. They analyse and compare the quality of the FMs designed using text mining with the FMs developed by experts. In Zhou *et al.* [6] authors discuss the importance of FM in identifying variability and commonality of SPL product. They use sentiment analysis to incorporate client's preferences.

4. PREDICTIVE SOFTWARE PRODUCT LINE TOOL (*p*-SPLIT)

In this section, we describe the proposed predictive framework. Figure 4 shows the architectural view of different modules of *p*-SPLIT.

4.1. Data Module

Data Module (DM) stores SPL configurations to the Product-Rep(ository). We acquired real-world data of an anonymous, multinational ERP-based SPL.[2] The essence of *p*-SPLIT is to give the developers decision support during the product configuration process. Moreover, configuration is a collaborative process, which involves multiple developer teams. Each team is solely responsible for the assigned configuration module. Keeping all these things in mind, it does not make sense to design a single dataset for a complete SPL product, because irrelevant configuration rules (or those related to other modules) being displayed on the developer's dashboard introduce more confusions and increase the complexity of the overall process. Therefore, we advocate the mining of SPL product configuration on a module by module basis. For *p*-SPLIT experiments, we acquired the configuration data of VM module from the configuration repository of our anonymous company. This data contained both types of configurations, i.e., consistent and inconsistent, stored in a text file. It contained a total of 178 different configurations.

[2]Due to a Nondisclosure Agreement, we cannot disclose the name of the company

4.2. Preprocessing Module

Before generating a complete dataset, we preprocessed and cleaned the individual configuration text files. For this, Preprocessing Module (PM) implemented programs in Java and C#.

PM checked the completeness of the data by analyzing the metadata and data stored in the files. The metadata contains the information about the SPL product configuration, including the number of features configured and the number of inconsistencies present in the configuration. PM imported the data from a text file to regenerate the configuration information and matched this information to the original metadata information to confirm the completeness and occurrence of missing features. In case, a data file was found to be faulty, PM used backup files. PM also checked the data for data inconsistencies by removing the duplicate entries of features.

PM also renamed the feature labels and combined the name of a feature with its constraint. For instance, for feature F_1, which is mandatory, the label is changed to M1. Similarly, the feature F_2, which is mandatory and include, is labeled as MI2. This renaming convention helped us to interpret better the extracted configuration rules.

After this, PM integrated these text files to produce a single data file. These configurations are imported into an Excel file, wherein each configuration was mapped to a row with a unique configuration ID. PM then transformed and encoded the data into binary data. A binary variable is created for each of the configuration features. After that, data reduction is applied and configuration ID column is removed from the dataset. For the given configuration with i features, PM maps them to a row with i columns, where bit strings (0 and 1) are used to encode the row, where 0 and 1 represent the deselection and selection of a feature, respectively. For instance, in the encoding "$010101, \cdots, 1$" F_2, F_4, F_6, and F_n are selected and all remaining features are deselected.

Furthermore, a column is introduced to represent the class label, which stores "Consistent" for the consistent configurations and "Inconsistent" for the inconsistent SPL product configurations. This dataset is stored in Product-Rep as **VM-DS** (VM DataSet) which is further divided into training, validation, and testing sets.

4.3. Predictive Analytics Module

Predictive Analytics Module (PAM) is the core module of *p*-SPLIT which implements RF DT to generate the decision support rules for SPL developers' team. It acquires the dataset of inconsistent and consistent product configurations from the DM (Product-Rep repository) to build a DT model. This model is validated on a testing set. The performance of models is compared in terms of their accuracy, precision, and recall. After the model validation, the decision rules are extracted from the model. Finally, these decision rules are passed to the Configuration Dashboard (CD). In PAM, we employed the RapidMiner tool [54] to estimate the predictions, and then visualized the results in C# language to provide decision support to the developers' teams.

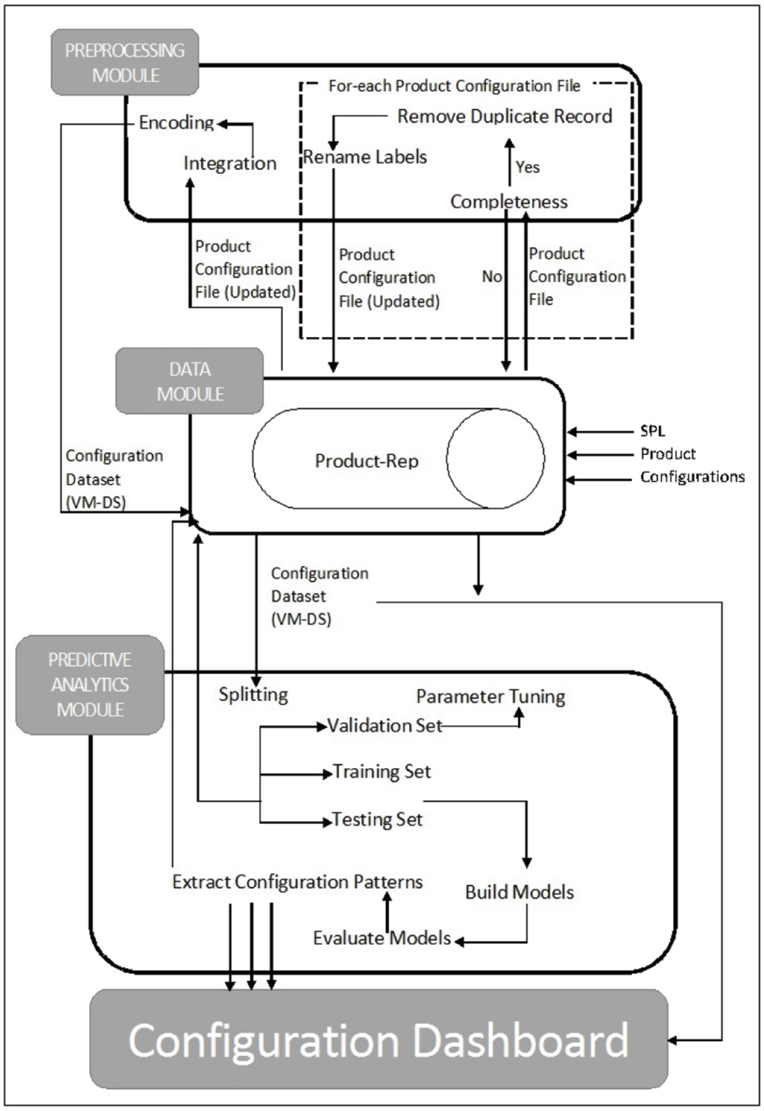

PREPROCESSING MODULE
For-each Product Configuration File
Remove Duplicate Record
Encoding
Integration
Rename Labels
Yes
Completeness
Product Configuration File (Updated)
Product Configuration File (Updated)
No
Product Configuration File
DATA MODULE
Product-Rep
SPL Product Configurations
Configuration Dataset (VM-DS)
Configuration Dataset (VM-DS)
PREDICTIVE ANALYTICS MODULE
Splitting
Parameter Tuning
Validation Set
Training Set
Testing Set
Extract Configuration Patterns
Build Models
Evaluate Models
Configuration Dashboard

Figure 4 Architecture of Predictive Software Product Line Tool (*p*-SPLIT): *p*redictive software product line tool.

4.3.1. Process design

The process flow of PAM starts with the reading of relevant dataset from the file. Then, the dataset is split into training, validation, and testing sets (a representative approach [55,56]). The primary parameters are tuned using the validation dataset. Using the training set with the tuned values of parameters, we then generate the RF models. These models are then applied on testing sets, and the performance for both sets is evaluated (recall that our performance measures are Accuracy, Kappa, AUC, Precision, and Recall). Finally, we generated the configuration rules from the trees which had better performance. We then hardcoded these into the CD. We performed all of these PAM experiments on a Windows 8 machine with Intel Core i7 CPU, 2.4 GHz processor, and 16GB of RAM.

4.3.2. Parameters tuning

The primary parameters of RF, such as the number of trees, criteria on which the attributes will be selected for tree splitting, the maximal depth of the tree, and prepruning were tuned according to the dataset and the nature of the problem. To determine the range of trees within a RF, we used the suggestions of [36]. We also tuned the maximal depth parameter with different bounds. To tune the feature selection criterion for splitting, we experimented PAM with information gain and gini index. We deselected the pruning, prepruning, and local seed selection options to obtain fully-grown DTs in RF. We allowed RF to guess the subset ratio to generate the trees.

To determine the range of trees within a RF, we used the suggestions of [36] and set the value to 70 which is a good balance between AUC, processing time, and memory usage. For the maximal depth parameter, we put no bound on the depth of the tree, i.e., a tree of maximum depth is generated. We used information gain as the feature selection criterion for splitting over gini index, as gini index targets continuous features while entropy targets features which occur in classes [57]. It should also be mentioned that in [58], the authors analyze the frequency of agreement/disagreement of the gini index and the information gain function. They found that the disagreement is only 2% and concluded that it is not possible to decide which one should be preferred. Therefore, our selection of information gain can also be attributed to our own preference and the previous experience. The optimal values of RF parameters which we selected are shown in Table 2.

4.4. Configuration Dashboard

CD in Figure 4 is an interface to provide the textual and graphical representation of the product under configuration. It also provides support to the developers on the basis of the decision rules, which are fetched from PAM. CD has the following features:

- It provides the current status of the configuration including: the number of features configured, number of inconsistencies in the current configuration along with the detail of each inconsistency type, and the Inconsistent features with their constraints.
- Configuration patterns, which lead to an inconsistent configuration.
- Configuration patterns, which lead to a consistent configuration.

We have implemented CD in C# programming language and it fetches all the relevant information from the DM (configuration repositories) and PAM.

5. RESULTS AND DISCUSSIONS

We ran PAM experiments based on the experimental methodology explained in the previous section. In this section, we discuss the results of these experiments.

Table 2 | PAM: optimal values for parameters.

Number of Trees	70
Criterion	Information Gain
Maximal Depth	No Bound

5.1. VM-DS Results and Discussion

In this subsection, we answer the research question RQ1. For this, we experimented VM-DS with the tuned value of the parameters (shown in Table 2). The dataset is splitted into two parts, i.e., training and testing. After that, we trained RF classifier on the training set and tested on the testing set. Finally, we analyze the performance of the model in terms of Accuracy, CE, Kappa, AUC, Precision, and Recall. Table 3 shows the result.

The accuracy of VM-DS model is 85% and CE is 14% with 0.66 kappa value and 0.978 AUC. The precision and recall of the VM-DS model are 81% and 73%, respectively.

Now, we discuss the possible role of PA through RF to solve the product configuration problem. PAM results show that the model generated from VM-DS has a good performance. Therefore, we used it to extract the configuration rules, which are further encoded to the CD. Figure 5 shows a snapshot of the RF generated with VM-DS. In this figure, the leaf labels represent either an inconsistent pattern (blue color) or a consistent pattern (red color). A complete red or blue leaf node represents a complete pattern for consistent or inconsistent configuration, respectively (i.e., no more features are needed to classify this particular consistency or inconsistency). Leaf nodes containing both red and blue color imply that more features are needed to acquire complete patterns. However, if one color has a larger frequency (and hence a larger size of the bar) than the other in this hybrid combination, then the former can be considered as the predicted class; RapidMiner has labeled these hybrid nodes accordingly.

Table 3 | PAM-results.

	VM-DS
Accuracy	85%
CE	14%
Kappa	0.66
AUC	0.978
Precision	81%
Recall	73%

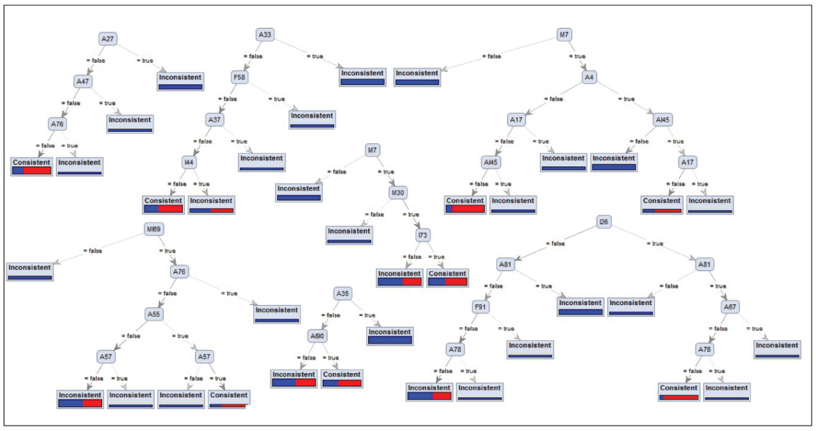


Figure 5 | A snapshot of the tree output by random forest algorithm.

To explain the extraction of the configuration rules and their usability, we randomly picked a DT from the RF.

Figure 6 shows a sub-tree of the RF tree shown in Figure 5, which presents three classification patterns for inconsistent configuration and two for the consistent one. The dynamics of the model are as follows:

- The selection of A37 leads to inconsistencies 75% of the time.
- Deselection of A37 along with M10 leads to inconsistencies 63% of the time.
- If M10 and I74 are selected, then not selecting A37 leads to consistent configuration 62% of the time.
- If M10 is selected, then not selecting A37, I74, and M7 leads to inconsistencies 57% of the time.
- The partial classification indicates that most of the configurations in our VM-DS-Comp data have a strong probability of turning out to be inconsistent.

These configurations rules along with their statistics and visuals are encoded to the CD, which helps developers during a product configuration and provides a runtime decision support.

We discuss the functionalities of CD with the help of a working example of an SPL product configuration. Figure 7 shows the configuration process of *VM-ERP* module of *CL-T-ERP*. The top left of Figure 7(A) shows the tracking information of *VM-ERP*, i.e., CL-T-ERP (product name) and T-ERP (SPL name), the top right shows the information of the features already configured within *VM-ERP*. The bottom of Figure 7(A) shows a list of the potential features which are not part of *VM-ERP*, but can still be selected. Figure 7 also shows an integrated support of PA-based CD. The bottom of Figure 7(B) shows how a runtime help is available to support the configuration decisions.

As the developer team selects F_{12} (through option button) to configure within *VM-ERP*, a configuration pattern shows the statistics of the selection and deselection of F_{12} on *VM-ERP* configurations. Figure 8 shows that the selection of F_{12} can lead to an inconsistent SPL product configuration and in case of F_{12} deselection, the developers should also need to pay attention to the configuration of F_{13}, F_6, and F_7. Figure 9 shows the configuration status of the SPL product *CL-T-ERP*. The center of the figure lists down the configured and unselected features for *CL-T-ERP* configuration, where *CL-T-ERP* contains F_1, F_2, and F_3; while F_8 and F_9 are unselected. The top right of the figure shows the graphical representation of the configured and unselected features for *CL-T-ERP* configuration. The top left of the figure (labeled as *Inconsistencies*) presents the details of inconsistencies by using an inconsistency operator. *CL-T-ERP* contains 40% inconsistent features with 20% alternative and 20% exclude inconsistencies, while it does not contain any type of mandatory and include inconsistencies.

CD also lists down the features which introduce the listed inconsistencies in the *Inconsistencies* combo boxes. The bottom left of the figure shows a visual representation of the inconsistencies. The top left of the figure (labeled as *Features*) shows the division of the configured features on the basis of their constraint types. *CL-T-ERP* configuration contains 30% mandatory features, 20% include and exclude features, and 30% alternative features. The bottom right of the figure presents graphs of the *CL-T-ERP* configured features distribution. The combo boxes of CD populate only the first 100 records to keep it efficient, while developers can customize combo boxes to show all of the records.

Based on this example and subsequent discussion, we can now answer our research question by stating that a PA-equipped CD can facilitate SPL developers teams in three ways: 1) By displaying patterns of feature selection leading to inconsistent product configurations, 2) By displaying patterns of feature selection leading to consistent product configurations, and 3) Predicting inconsistencies in products that are currently under configuration.

5.2. Subjective Evaluation and Benefits of *p*-SPLIT

To acquire subjective feedback for *p*-SPLIT, we implemented *p*-SPLIT in our client company whose FM and datasets were used to run our experiments. Initially, we developed a testing environment to configure a medium-scale ERP product for a team of ten developers (comprising four junior and six senior developers). We setup ten test servers and a single database server, where one test server was assigned to every developer. We connected all test servers to the database server for sharing SPL repositories and equipped each with *p*-SPLIT interface. We populated the database server with configuration repositories of *p*-SPLIT. We also populated the *p*-SPLIT repositories with the test FM data for medium-scale configuration. We started the testing process with domain engineering of a medium-scaled FM. After that, we configured a product for an exemplary client. runtime decision support was available throughout the configuration process. After the successful execution of *p*-SPLIT in the testing environment, we acquired subjective feedback from the developers involved in the test configuration.

We acquired this feedback through a subjective questionnaire based on standard guidelines [59]. The developers provided feedback for *p*-SPLIT results on a scale of 1 (strongly disagree) to 5 (strongly agree) for the following four questions:

- Q1: The statistics displayed on CD are helpful.
- Q2: The PAM results, displayed on CD, provide an appropriate runtime feature selection and deselection support.
- Q3: The decision support provided by CD is efficient as compared to the manual configurations.
- Q4: The decision support provided by CD has a practical applicability to the business domain.

We acquired feedback from all the ten developers and calculated the average response for each question. Q3 received an average of 5, i.e., all developers strongly agreed that the decision support by *p*-SPLIT makes the configuration process more efficient as compared to traditional method. Q1 and Q2 received an average of 3.8, while Q4 received an average of 4. Although we agree that these results are acquired for only a single evaluation in a limited setting, they do indicate that *p*-SPLIT has the potential to provide the relevant decision support for SPL developers.

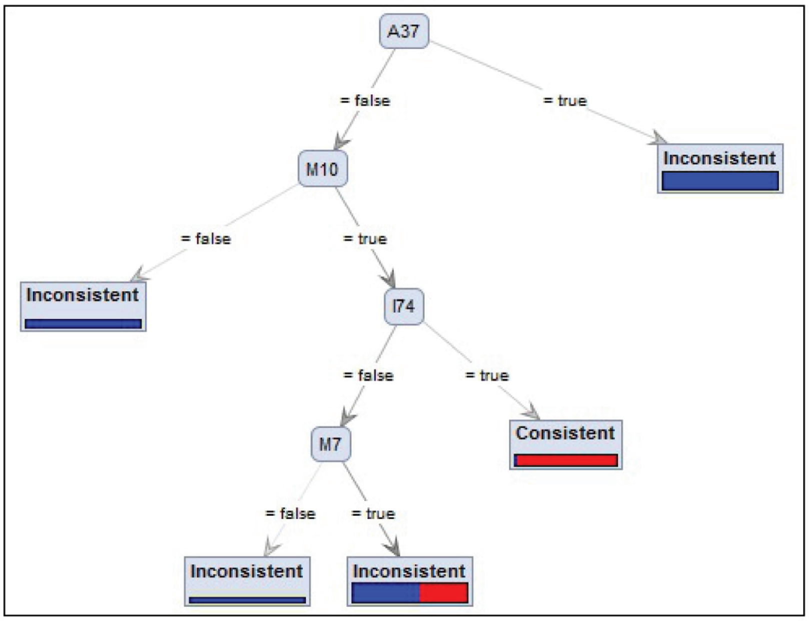
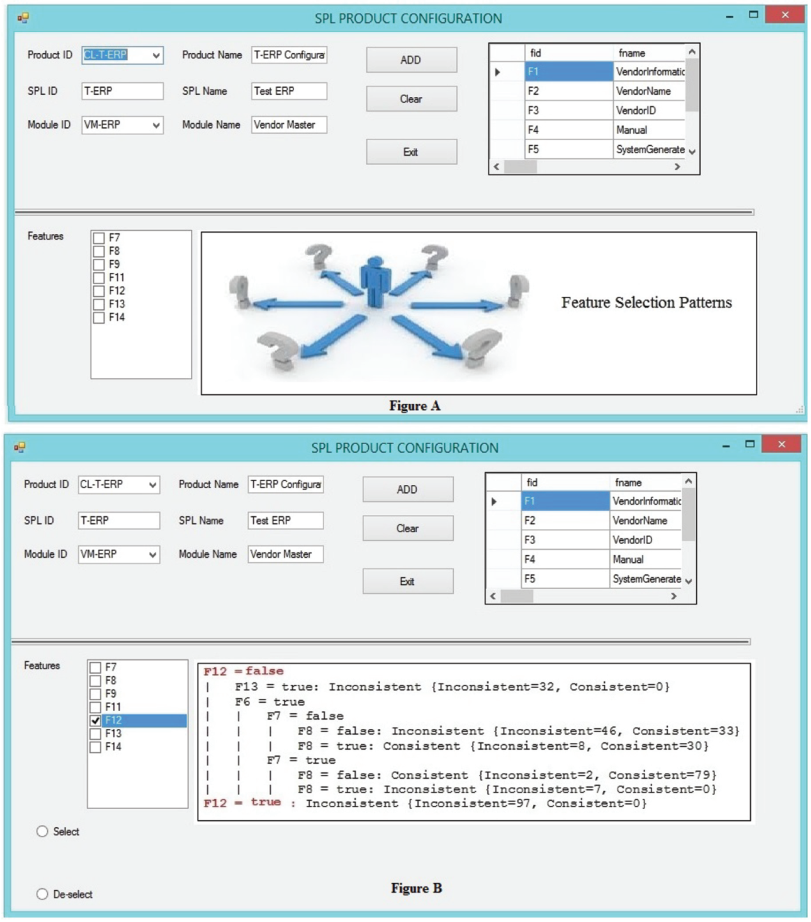


Figure 6 A sub-tree of the random forest tree shown in Figure 5.


SPL PRODUCT CONFIGURATION
ADD
Clear
Exit
Features
Feature Selection Patterns
Figure A
SPL PRODUCT CONFIGURATION
ADD
Clear
Exit
Features
Select
De-select
Figure B

Product ID: CL-T-ERP
SPL ID: T-ERP
Module ID: VM-ERP
Product Name: T-ERP Configura
SPL Name: Test ERP
Module Name: Vendor Master

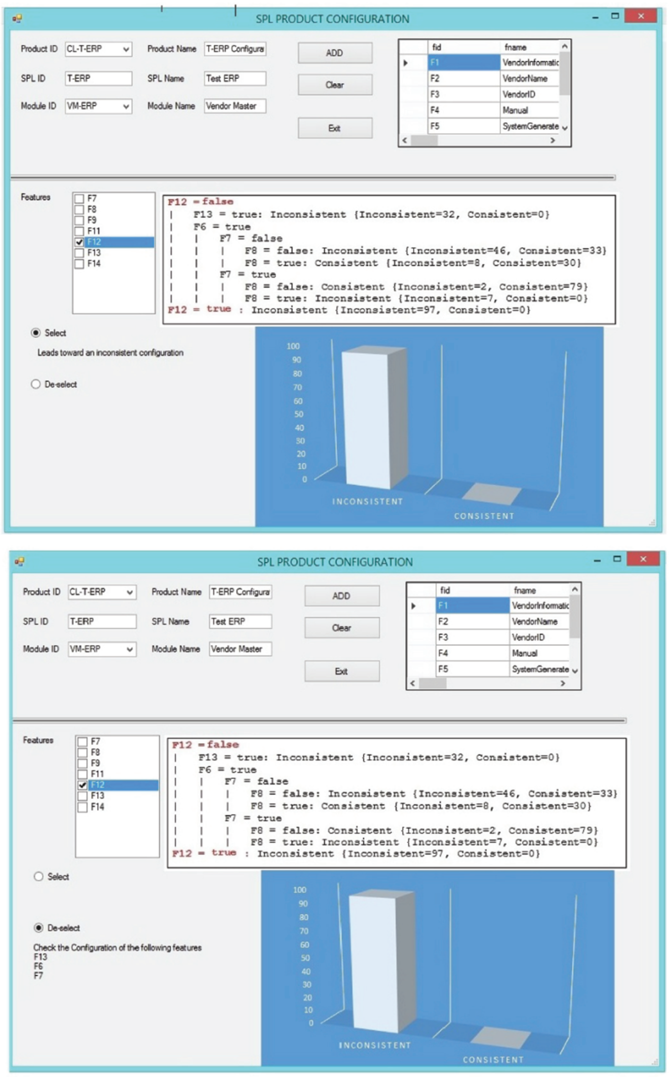
fid	fname
F1	VendorInformatic
F2	VendorName
F3	VendorID
F4	Manual
F5	SystemGenerate

Features: F7, F8, F9, F11, F12, F13, F14

Features: F7, F8, F9, F11, [✓ F12], F13, F14

```
F12 =false
|   F13 = true: Inconsistent {Inconsistent=32, Consistent=0}
|   F6 = true
|   |   F7 = false
|   |   |   F8 = false: Inconsistent {Inconsistent=46, Consistent=33}
|   |   |   F8 = true: Consistent {Inconsistent=8, Consistent=30}
|   |   F7 = true
|   |   |   F8 = false: Consistent {Inconsistent=2, Consistent=79}
|   |   |   F8 = true: Inconsistent {Inconsistent=7, Consistent=0}
F12 = true : Inconsistent {Inconsistent=97, Consistent=0}
```

Figure 7 Software product line configuration dashboard (SPL CD) of predictive software product line tool (*p*-SPLIT).



Product ID	CL-T-ERP
Product Name	T-ERP Configura
SPL ID	T-ERP
SPL Name	Test ERP
Module ID	VM-ERP
Module Name	Vendor Master

fid	fname
F1	VendorInformatic
F2	VendorName
F3	VendorID
F4	Manual
F5	SystemGenerate

Features: ☐ F7, ☐ F8, ☐ F9, ☐ F11, ☑ F12, ☐ F13, ☐ F14

```
F12 =false
|    F13 = true: Inconsistent {Inconsistent=32, Consistent=0}
|    F6 = true
|    |    F7 = false
|    |    |    F8 = false: Inconsistent {Inconsistent=46, Consistent=33}
|    |    |    F8 = true: Consistent {Inconsistent=8, Consistent=30}
|    |    F7 = true
|    |    |    F8 = false: Consistent {Inconsistent=2, Consistent=79}
|    |    |    F8 = true: Inconsistent {Inconsistent=7, Consistent=0}
F12 = true : Inconsistent {Inconsistent=97, Consistent=0}
```

● Select

Leads toward an inconsistent configuration

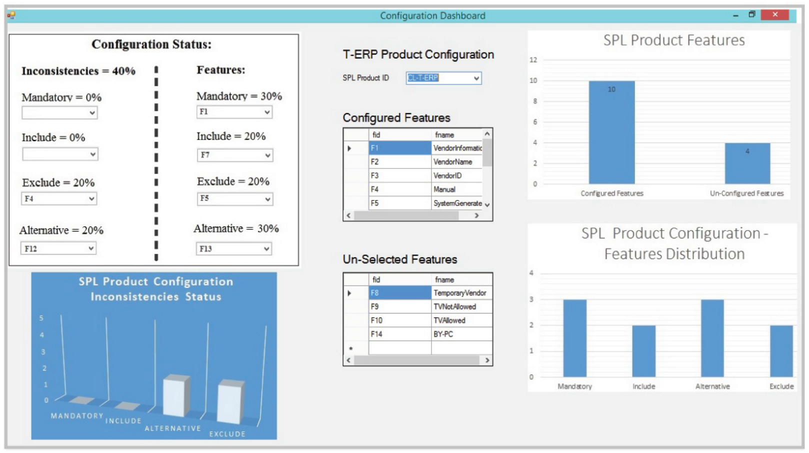
○ De-select

○ Select

● De-select

Check the Configuration of the following features
F13
F6
F7

Figure 8 | Detailed configuration dashboard (CD) of predictive software product line tool (*p*-SPLIT).



Configuration Status:

Inconsistencies = 40%	Features:
Mandatory = 0%	Mandatory = 30% F1
Include = 0%	Include = 20% F7
Exclude = 20% F4	Exclude = 20% F5
Alternative = 20% F12	Alternative = 30% F13

T-ERP Product Configuration

SPL Product ID: CL-T-ERP

Configured Features

fid	fname
F1	VendorInformatic
F2	VendorName
F3	VendorID
F4	Manual
F5	SystemGenerate

Un-Selected Features

fid	fname
F8	TemporaryVendor
F9	TVNotAllowed
F10	TVAllowed
F14	BY-PC

Figure 9 | Product details shown in configuration dashboard.

Besides this subjective evaluation, we present a comparison of two widely used industrial tools with *p*-SPLIT to highlight the advantages of proposed solution. Pure::variants [60] and Gears [61] have widespread applications as compared to other tools [62,63].

Table 4 A Comparison of state-of-the-art gears and pure::variants industrial tools with *p*-SPLIT.

	pure::variants	**Gears**	***p-SPLIT***
Generate Pattern of Inconsistent Features	X	X	✓
Generate Pattern of Consistent Features	X	X	✓
Predicting Inconsistencies	X	X	✓
Identify Inconsistencies	✓	✓	✓
Resolve Inconsistencies	✓	✓	X

Table 4 shows that *p*-SPLIT can facilitate SPL developers teams in three ways:

- Classifying patterns of feature selection leading to inconsistent FM.
- Classifying patterns of feature selection leading to consistent FM
- Predicting inconsistencies in FM that are currently under configuration.

p-SPLIT also provides current status of the configuration including:

- Number of features configured.
- Number of inconsistencies in the current configuration with the detail of each inconsistency type.
- Inconsistent features with their constraints.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we present a novel technology for the SPL business domain called *p*-SPLIT, which uses PA to address the SPL product configuration issues by providing decision support to SPL developers' team at runtime. *p*-SPLIT provides this runtime by offering a CD, which helps the developers using the configuration rules generated by PA. *p*-SPLIT classifies the pattern of configurations of feature selection leading to inconsistent and consistent product configurations. It also predicts the inconsistencies of FM which are currently under configuration by showing them on CD.

As a future work, we intend to expand our experiments with large-scale datasets (big data) and equip the CD with relevant inter-module configurations patterns. Work on the the inter-operability of *p*-SPLIT with other industrial tools such as Gears and Pure-variants is another future direction. We are also planning to share *p*-SPLIT through a web interface which can be integrated with SPLOT and BeTTy and made freely available for academic and research purpose.

CONFLICTS OF INTEREST

The authors declares that they have no conflict of interest.

REFERENCES

[1] H. Hartmann, Software product line engineering for consumer electronics, keeping up with the speed of innovation, PhD thesis, University of Groningen, Groningen, Netherlands, 2015.

[2] J.-M. Horcas, M. Pinto, L. Fuentes, Software product line engineering: a practical experience, in Proceeding of 23rd International Systems and Software Product Line Conference, Paris, France, 2019, vol. A, pp. 164–176.

[3] L.M. Northrop, P.C. Clements, A Framework for Software Product Line Practice, version 5.0, Carnegie Mellon University, USA, 2012.

[4] H. Chemingui, I. Gam, R. Mazo, C. Salinesi, H.B. Ghezala, Product line configuration meets process mining, Procedia Comput. Sci. 164 (2019), 199–210.

[5] J. White, D. Benavides, D.C. Schmidt, P. Trinidad, B. Dougherty, A. Ruiz-Cortes, Automated diagnosis of feature model configurations, J. Syst. Softw. 83 (2010), 1094–1107.

[6] F. Zhou, J. Roger Jiao, X.J. Yang, B. Lei, Augmenting feature model through customer preference mining by hybrid sentiment analysis, Expert Syst. Appl. 89 (2017), 306–317.

[7] U. Afzal, T. Mahmood, A.H. Khan, S. Jan, R.U. Rasool, A.M. Qamar, R. Ullah Khan, Feature selection optimization in software product lines, IEEE Access. 8 (2020), 160231–160250.

[8] U. Afzal, T. Mahmood, Z. Shaikh, Intelligent software product line configurations: a literature review, Comput. Stand. Interface. 48 (2016), 30–48.

[9] M. Bhushan, A. Negi, P. Samant, S. Goel, A. Kumar, A classification and systematic review of product line feature model defects, Softw. Qual. J. 28 (2020), 1507–1550.

[10] T. Buchmann, F. Schwagerl, A repair-oriented approach to product consistency in product lines using negative variability, SICS Softw. Intensive Cyber Phys. Syst. 34 (2019), 17–33.

[11] A. Hubaux, Feature-based configuration: collaborative, dependable, and controlled, PhD thesis, Department of Computer Science, University of Namur, Namur, Belgium, 2012.

[12] M. Marques, J. Simmonds, P.O. Rossel, M. Cecilia Bastarrica, Software product line evolution: a systematic literature review, Inf. Softw. Technol. 105 (2019), 190–208.

[13] M. Bhushan, S. Goel, K. Kaur, Analyzing inconsistencies in software product lines using an ontological rule-based approach, J. Syst. Softw. 137 (2018), 605–617.

[14] D.C. Botero, A Generic method for assembling software product line components, PhD thesis, Departamento de Ciencias de la Computacion y la Decision, Universidad Nacional de Colombia, Medellín, Colombia, 2020.

[15] I. Dávid, E. Syriani, C. Verbrugge, D. Buchs, D. Blouin, A. Cicchetti, K. Vanherpen, Towards inconsistency tolerance by quantification of semantic inconsistencies, in COMMitMDE@ MoDELS, First International Workshop on Collaborative Modelling in MDE COMMITMDE, Saint Malo, France 2016, pp. 35–44.

[16] R. Flores, C. Krueger, P. Clements, Mega-scale product line engineering at general motors, in Proceedings of the 16th

International Software Product Line Conference (SPLC), Salvador, Brazil, 2012, vol. 1, pp. 259–268.

[17] J. Van Gurp, C. Prehofer, From spls to open, compositional platforms, in Combining the Advantages of Product Lines and Open Source, volume 8142 of Dagstuhl Seminar, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Saarbrucken, Germany.

[18] N. Niu, J. Savolainen, Y. Yu, Variability modeling for product line viewpoints integration, in 2010 IEEE 34th Annual Computer Software and Applications Conference, Seoul, South Korea, 2010, pp. 337–346.

[19] K. Pohl, G. Böckle, F.J. van der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques, vol. 10, Springer-Verlag, Berlin, Germany, 2005.

[20] S. Aqeel Safdar, H. Lu, T. Yue, S. Ali, Mining cross product line rules with multi-objective search and machine learning, in Proceeding of the Genetic and Evolutionary Computation Conference, Berlin, Germany, 2017, pp. 1319–1326.

[21] X. Tërnava, P. Collet, Early consistency checking between specification and implementation variabilities, in Proceeding of 21st International Systems and Software Product Line Conference, Sevilla, Spain, 2017, vol. A, pp. 29–38.

[22] C. Thao, A configuration management system for software product lines, MS thesis, University of Wisconsin Milwaukee, Milwaukee, WI, USA, 2012.

[23] Y. Zhan, K.H. Tan, B. Huo, Bridging customer knowledge to innovative product development: a data mining approach, Int. J. Prod. Res. 57 (2019), 6335–6350.

[24] S. Russell, P. Norvig, Artificial Intelligence: a Modern Approach, third ed., Pearson, USA, 2009.

[25] M. Noorian, A. Ensan, E. Bagheri, H. Boley, Y. Biletskiy, Feature model debugging based on description logic reasoning, Proc. DMS. 11 (2011), 158–164.

[26] A.O. Elfaki, S. Phon-Amnuaisuk, C. Kuan Ho, Knowledge based method to validate feature models, in Proceeding of Software Product Line Conference (SPLC), Limerick, Ireland, 2008, vol. 2, pp. 217–225.

[27] P. Trinidad, A.R. Cortés, Abductive reasoning and automated analysis of feature models: How are they connected?, in Third International Workshop on Variability Modelling of Software-Intensive Systems, Seville, Spain, 2009, vol. 9, pp.145–153.

[28] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later: a literature review, Inf. Syst. 35 (2010), 615–636.

[29] E. Alpaydn, Introduction to Machine Learning (Adaptive Computation and Machine Learning), MIT Press, USA, 2004.

[30] S. Philpott, Advanced Analytics: Unlocking the Power of Insight, IBM, USA, 2010.

[31] E. Siegel, Predictive Analytics: the Power to Predict who will Click, Buy, Lie, or Die, John Wiley & Sons, USA, 2013.

[32] L. Breiman, Random forests, Mach. Learn. 45 (2001), 5–32.

[33] M. Fernández-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems, J. Mach. Learn. Res. 15 (2014), 3133–3181.

[34] A. Liaw, M. Wiener, Classification and regression by randomforest, R News. 2 (2002), 18–22.

[35] K.L. Lunetta, L.B. Hayward, J. Segal, P. Van Eerdewegh, Screening large-scale association study data: exploiting interactions using random forests, BMC Genet. 5 (2004), 32.

[36] T.M. Oshiro, P.S. Perez, J.A. Baranauskas, How many trees in a random forest?, in Proceeding of International Workshop on Machine Learning and Data Mining in Pattern Recognition, Springer, Berlin, Germany, 2012, pp. 154–168.

[37] K. Czarnecki, S. Helsen, U. Eisenecker, Staged configuration using feature models, in Proceeding of Software Product Line Conference (SPLC), Boston, MA, USA, 2004, pp. 266–283.

[38] E. Monk, B. Wagner, Concepts in Enterprise Resource Planning, Cengage Learning, USA, 2012.

[39] C. Bishop, Pattern Recognition and Machine Learning: Information Science and Statistics, first ed., Springer-Verlag, New York, NY, USA, 2006.

[40] J. McClave, T. Sincich, Statistics, twelth ed., Pearson, UK, 2012.

[41] R. Yera, L. Martinez, Fuzzy tools in recommender systems: a survey, Int. J. Comput. Intell. Syst. 10 (2017), 776–803.

[42] L. Breiman, Bagging predictors, Mach. Learn. 24 (1996), 123–140.

[43] R.E. Schapire, Y. Freund, P. Bartlett, W. Sun Lee, Boosting the margin: a new explanation for the effectiveness of voting methods, Ann. Stat. 26 (1998), 1651–1686.

[44] C. Kastner, A. Dreiling, K. Ostermann, Variability mining: consistent semi automatic detection of product line features, IEEE Trans. Softw. Eng. 40 (2014), 67–82.

[45] N. Niu, S. Easterbrook, On-demand cluster analysis for product line functional requirements, in 2008 12th International Software Product Line Conference (SPLC), Limerick, Ireland, 2008, pp. 87–96.

[46] A. Lora-Michiels, A method based on association rules to construct product line model, in Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria, 2010.

[47] M. Gardner, J. Bieker, Data mining solves tough semiconductor manufacturing problems, in Proceeding of Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2000, pp. 376–383.

[48] S.R. Faulk, Product line requirements specification (PRS): an approach and case study, in Proceeding of Fifth IEEE International Symposium on Requirements Engineering, IEEE, Toronto, Canada, 2001, pp. 48–55.

[49] J. Maazoun, N. Bouassida, H. Ben Abdallah, A new approach mining the spl feature model and design from product variants, Ada User J. 39 (2018), 37–47.

[50] J.R. Jiao, T.W. Simpson, Z. Siddique, Product family design and platform-based product development: a state of the art review, J. Intell. Manuf. 18 (2007), 5–29.

[51] M.A.P.M. Lejeune, Measuring the impact of data mining on churn management, Internet Res. 11 (2001), 375–387.

[52] C.T. Su, Y.H. Chen, D.Y. Sha, Linking innovative product development with customer knowledge: a data mining approach, Technovation. 26 (2006), 784–795.

[53] J.H. Heinrichs, J.-S. Lim, Integrating web-based data mining tools with business models for knowledge management, Decis. Support Syst. 35 (2003), 103–112.

[54] RapidMiner: Best data science and machine learning platform, 2020. https://rapidminer.com/

[55] R.S. Michalski, J.G. Carbonell, T.M. Mitchell, Machine Learning: an Artificial Intelligence Approach, Springer Science & Business Media, Berlin, Heidelberg, Germany, 2013.

[56] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann, USA, 2005.

[57] G. Sieling, Decision trees: "gini" vs. "entropy" criteria, 2014. https://www.garysieling.com/blog/sklearn-gini-vsentropy-criteria/.

[58] L.E. Raileanu, K. Stoffel, Theoretical comparison between the gini index and information gain criteria, Ann. Math. Artif. Intell. 41 (2004), 77–93.

[59] I. Brace, Questionnaire Design: How to Plan, Structure and Write Survey Material for Effective Market Research, Kogan Page Publishers, London, England, 2008.

[60] PureSystems, Variant management, 2014. https://www.pure-systems.com/.

[61] BigLever, Gears, 2020. http://www.biglever.com/solution/product.html.

[62] F. Qudus Khan, S. Musa, G. Tsaramirsis, S.T. Bakhsh, A study: selection of model metamodel and spl tools for the verification of software product lines, Int. J. Inf. Technol. 9 (2017), 353–362.

[63] J.A. Pereira, K. Constantino, E. Figueiredo, A systematic literature review of software product line management tools, in Proceeding of International Conference on Software Reuse, Miami, FL, USA, 2015, pp. 73–89.