Software and Systems Modeling manuscript No.
(will be inserted by the editor)

Enhancing software model encoding for feature location approaches based on machine learning techniques

Ana C. Marcén[1,2], Francisca Pérez[1], Óscar Pastor[2], Carlos Cetina[1]

[1] SVIT Research Group, Universidad San Jorge, Autovía A-23 Zaragoza-Huesca Km.299, 50830, Zaragoza, Spain
e-mail: acmarcen@usj.es.com, mfperez@usj.es, ccetina@usj.es

[2] Centro de Investigación en Métodos de Producción de Software, Universitat Politècnica de València, Camino de Vera S/N, 46022, Valencia, Spain
e-mail: opastor@pros.upv.es

Received: date / Accepted: date

Abstract Feature location is one of the main activities performed during software evolution. In our previous works, we proposed an approach for feature location in models based on machine learning, providing evidence that machine learning techniques can obtain better results than other retrieval techniques for feature location in models. However, to apply machine learning techniques optimally, the design of an encoding is essential to be able to identify the best realization of a feature.

In this work, we present more thorough research about software model encoding for feature location approaches based on machine learning. As part of this study, we have provided two new software model encodings and compared them with the source encoding. The first proposed encoding is an extension of the source encoding to take advantage of not only the main concepts and relations of a domain but also the properties of these concepts and relations. The second proposed encoding is inspired by the characteristics used in benchmark datasets for research on Learning to Rank. Afterward, the new encodings are used to compare three different machine learning techniques (RankBoost, Feedforward Neural Network, and Recurrent Neural Network). The study also considers whether a domain-independent encoding such as the ones proposed in this work can outperform an encoding that is specifically designed to exploit human experience and domain knowledge. Furthermore, the results of the best encoding and the best machine learning technique were compared to two traditional approaches that have been widely for feature location as well as for traceability link recovery and bug localization.

The evaluation is based on two real-world case studies, one in the railway domain and the other in the induction hob domain. An approach for feature location in models evaluates these case studies with the different encodings and machine learning techniques. The results show that when using the second proposed encoding and RankBoost, the approach outperforms the results of the other encodings and machine learning techniques and the results of the traditional approaches. Specifically, the approach achieved the best results for all the performance indicators, providing a mean precision value of 90.11%, a recall value of 86.20%, a F-measure value of 87.22%, and a MCC value of 0.87. The statistical analysis of the results shows that this approach significantly improves the results and increases the magnitude of the improvement. The promising results of this work can serve as a starting point towards the use of machine learning techniques in other engineering tasks with software models, such as traceability or bug location.

Key words Software Models; Feature Location; Machine Learning; Learning to Rank; Neural Networks; Encoding

1 Introduction

Feature location is known as the process of finding the set of software artifacts that realize a specific functionality of a software system. Feature location is one of the main activities performed during software evolution [37] and up to 80% of a system's lifetime is spent on the maintenance and evolution of the system [54]. Due to the importance of feature location, researchers have proposed a number of approaches to improve developers' effectiveness in locating features, which are largely based on source code [79,25,5,19,34] and are less frequently based on other artifacts, such as models [89,42, 92,93,63,28].

In our previous works [60,58,59], we proposed an approach for feature location in models based on machine learning (FLiM-ML), providing evidence that machine

learning techniques can be applicable to feature location in models. The FLiM-ML approach obtained better results than an approach based on latent semantic indexing, which is the most commonly used information retrieval technique and which provided the best results for feature location in models [72].

To do this, the design of an encoding is essential in order to apply machine learning techniques optimally. An encoding is the characterization or representation of the object being observed so that this object can be understood and used by machine learning techniques. For example, in weather forecasting, the weather of a day is characterized by means of several measures (e.g., minimum and maximum temperature, wind speed, or relative humidity), which are used by machine learning techniques to predict the weather for the next day. In feature location in models, the models can have very different types of elements, structures, purposes, and they can even belong to different domains. There is not an evident or single way to characterized them.

Therefore, after demonstrating the success of the machine learning-based approach for feature location in models, encoding became the main point of interest in order to improve the results of the approach. The contribution of this paper focuses on more thorough research on encoding in order to provide other alternatives for encoding model fragments and feature descriptions and also to compare the results of the different encodings using the FLiM-ML approach.

Currently, most works on feature location focus on locating features in source code using information retrieval techniques, such as latent semantic indexing. However, some recent works have presented approaches for feature location in models instead of source code. Some of these works have even taken one step further in feature location in models, using machine learning techniques to improve the results of traditional information retrieval techniques [60].

We propose two new encodings, and we use the encoding proposed in [60] as baseline. The first one (extended encoding) is an extension of the source encoding that takes advantage of the main concepts and relations of a domain (as the source encoding does) and the properties of these concepts and relations. This provides more details, which may benefit the machine learning process. The second one (mapped encoding) is inspired by the characteristics used in benchmark datasets for research on Learning to Rank [76,75,74]. Although the artifacts encoded in benchmark datasets are neither feature descriptions nor model fragments, this proposed encoding adapts the characteristics that have been tested and used in other research communities for years to software models.

The evaluation of both the source encoding and the two new encodings is based on two industrial case studies with 1800 test cases and 108 test cases, respectively. The first case study was provided by CAF[1], a worldwide provider of railway solutions. The second one was provided by BSH[2], one of the largest manufacturers of home appliances in Europe. Although this work has evaluated the encodings in two specific domains, the encodings are designed to be general and independent of the domain so that they can be applied in other domains. Moreover, for the evaluation to be fair, the FLiM-ML approach was set up to perform feature selection and to tune parameters for each one of the evaluated encodings. Feature selection simplifies the classifiers and reduces overfitting, so it is widely extended in mining and machine learning applications [36]. Since parameter tuning is often more important than the choice of a machine learning technique [50], the FLiM-ML approach is tuned to provide the best performance of the classifier for each encoding.

Based on the case studies and taking into account feature selection and parameter tuning, we first compared the results of the approach using the source encoding and the two new proposed encodings: the extended encoding and the mapped encoding. The results show that the FLiM-ML approach achieved the best results using the mapped encoding. The statistical analysis shows that there are no significant differences between the results of the approach using the source encoding and the extended encoding. In contrast, the statistical analysis shows that the mapped encoding significantly outperforms the baseline.

Second, we used the encodings to evaluate three different machine learning techniques. Specifically, we evaluated a Learning to Rank algorithm (RankBoost) and two neural networks (a Feedforward Neural Network and a Recurrent Neural Network). The results show that the FLiM-ML approach achieved the best results using RankBoost and the mapped encoding.

Third, we compared the previous results to two encodings designed by domain experts. The results provide evidence that a domain-independent encoding, such as the mapped encoding, can outperform the results obtained using encodings that are specifically designed to exploit human experience and domain knowledge.

Finally, the best results were compared with two traditional approaches, which have been widely applied for feature location as well as for traceability link recovery and bug localization [90]. The first one [84] is a Linguistic Rule-Based (Linguistic-baseline) approach that is based on parts-of-speech tagging and traceability rules. The second one [22,55] is an Information Retrieval (IR-baseline) approach that is based on latent semantic indexing and singular value decomposition. The mapped encoding with RankBoost outperforms the results of the two approaches.

This work provides evidence that the results of the machine learning techniques can be improved even more

[1] `https://www.caf.net/en`

[2] `https://www.bsh-group.com/`

through the enhancement of the encoding. The successful results of this work open the door to exploring the use of machine learning techniques for other software problems in models, such as requirements traceability [49] or bug location [2].

The remainder of this paper is structured as follows: Section 2 provides background and motivation for feature location in models. Section 3 presents both the source encoding and the two new proposed encodings. Section 4 details the means used to evaluate our work and the results of the evaluation. Section 5 analyzes the statistical significance of the obtained results. Section 6 discusses our approach and the obtained results. Section 7 describes the threats to the validity of our work. Section 8 introduces the existing works that are related to our work. Finally, Section 9 concludes the paper.

2 Background and motivation

In model driven engineering, models are the main software artifacts. Models raise the abstraction level using concepts that are much less bound to the underlying implementation and technology and are much closer to the problem domain [12]. The practice of model driven engineering has proven to increase efficiency and effectiveness in software development [12]. In fact, in industrial contexts, fostering modeling efforts brings benefits that improve productivity, while ensuring quality and performance [12].

Therefore, in a model-driven industrial context, companies tend to have a myriad of products with large and complex models behind them [71]. Since the software engineers must spend great amounts of time and effort in locating the model elements that must be maintained or evolved, an approach that automatically retrieves model fragments is greatly needed [71]. Machine learning is an attractive solution for reducing maintenance costs and exploiting resources. Companies that have been developing software systems for a long time have gathered a large amount of knowledge and experience (e.g., features that they manually locate) and machine learning techniques are the key to exploiting these resources by automating retrieval and reducing costs.

In fact, machine learning techniques can be an advantage for the exploitation not only of the material resources, but also of human resources. For example, if a newly hired engineer has to develop the door control for a train, he will have to start the development from scratch. As a newcomer, he does not know the company's models. Therefore, this job will require a strong effort due to his lack of experience. Following the SCRUM methodology, a large working set (Epic) can be divided into specific tasks (Stories). Door control development can be broken down into: (1) finding a model fragment similar to the one you want to develop and (2) modifying it based on new requirements. Since the company's models are unknown to the engineer, locating or identifying a similar fragment will be like looking for a needle in a haystack. A machine learning-based approach, like FLiM-ML, can automatically locate the model fragments for the engineer. This allows that the engineer focuses on modifying an existing fragment, rather than creating it from scratch. Therefore, a job, that seemed impossible for a new developer, can be tackled based on the exploitation of previous resources through machine learning.

The following section presents a detailed description of the feature location problem in models and an overview of the problem regarding machine learning.

2.1 Feature location in models

Feature location is known as the process of finding the set of software artifacts that realizes a specific functionality of a software system. The term 'feature' refers to a specific functionality of a product. In feature location in models, this functionality is realized by a model fragment. A model fragment is composed of one or more elements of the product model, which may or may not be connected with each other. The elements of a model fragment may or may not be associated with the functionality. Therefore, the goal is to identify the model fragment that contains the model elements that are associated with a specific functionality.

Fig. 1 depicts an example that is taken from a real-world train. Fig. 1 (left) shows the inputs for feature location, a feature description, and a product model. The feature descriptions used in this work are defined using natural language and the product models are MOF compliant models that were created with a domain specific language for trains called Train Control and Management Language (TCML[3]). TCML includes both the structural and the behavioral design. In fact, the whole source code, that controls all the equipment of a train, is obtained from this domain specific language. It has the expressiveness required to describe the interaction between the main pieces of a train and to specify non-functional aspects that are related to regulation.

In the example of Fig. 1, the description corresponds to a "car door state signaling" feature. The model has two separate buttons that control the state of the doors installed in the cabin and the cars of a train. When the buttons are pushed, the state of the doors is modified and the LED of the buttons changes to indicate whether the doors are open, closed, or blocked. To open *Door1*, a special permission is necessary to access the cabin of the train. Therefore, the composition, the cabin, the cars, the doors, the buttons, and the connections among them are the model elements. The access, the state, the pushed condition of the buttons, and the LED are the element properties.

[3] Learn more of TCML at: https://youtu.be/Ypcl2evEQB8

Fig. 1 Example of a TCML model and model fragment

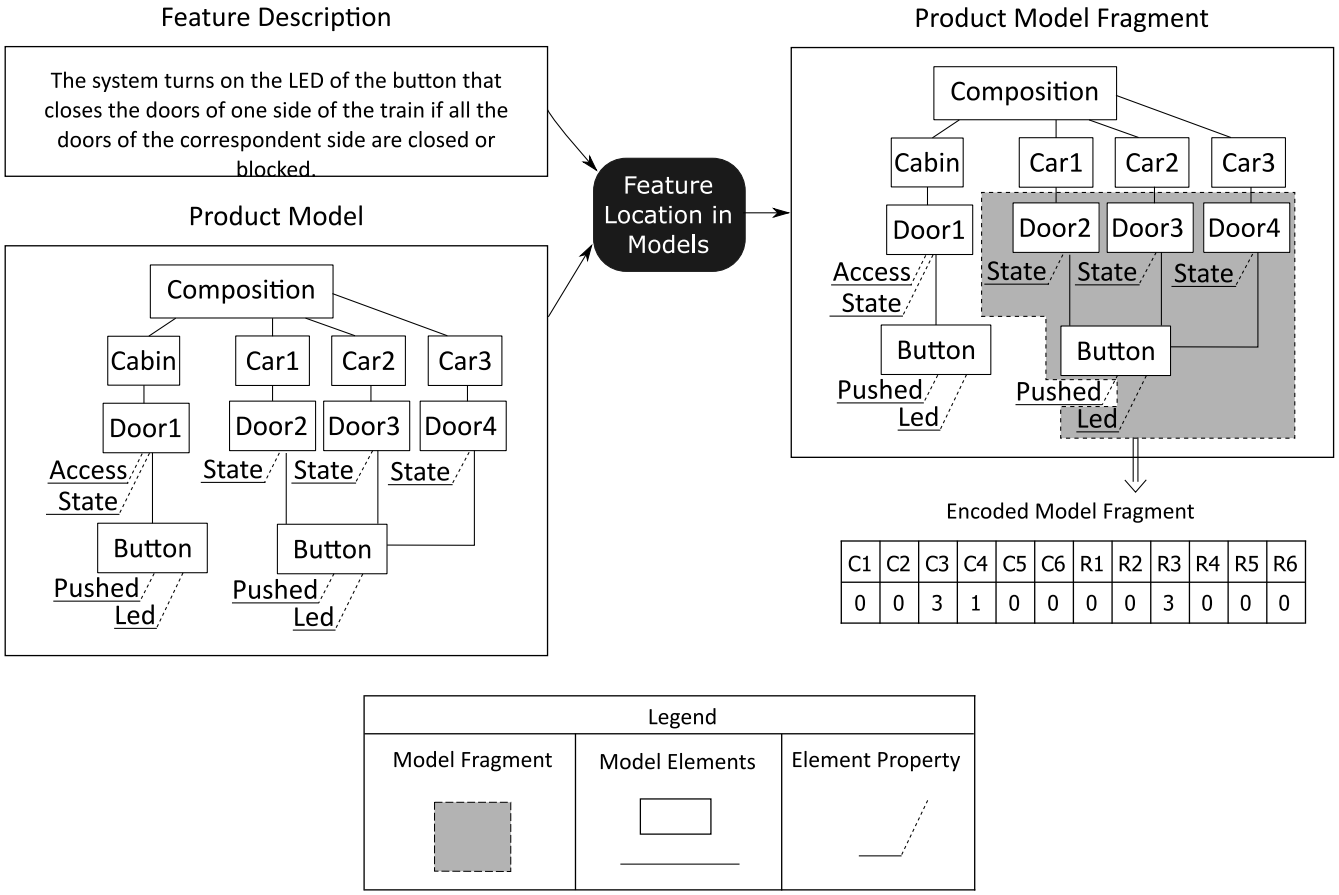


Fig. 1 (right) shows an example of a model fragment, which is obtained as output from locating the "car door state signaling" feature. This model fragment is shaded in gray with dashed lines and contains the model elements that are relevant for the described feature. In this example, the model fragment includes the three doors that are installed in the cars, the state of these doors, the button that controls these doors, the LED that indicates the state of the doors, and the connections between the doors and the button. The occurrences of these elements are used to encode the model fragment into a vector. For example, the number of doors in the model fragment (three) is the value for the position *C3* in the vector and the number of buttons (one) is the value for the position *C4* in the vector.

Although the example in Fig. 1 makes manual feature location in models look easy, it becomes very complex in the models of our industrial partner. The real models of our industrial partner CAF have more than 650 model elements, and each element has about 15 properties. This example shows a door control scenario with only 23 model elements and 9 properties in order to help better understand the feature location problem in models. However, it does not reflect the complexity of the problem.

Suppose we ask the domain expert to manually locate the model elements that correspond to the 10 features of our case study provided by CAF. To locate one of these features, the domain expert evaluates the model elements of the 20 models. Since each model has about 650 model elements, at least 13,000 model elements should be evaluated. To assess a model element, it is reasonable to consider its properties. In the case study, each element has about 15 properties, so about 195,000 properties of model elements should be considered. Assuming that a domain expert only needs 1 second to consider a property of a model element, the domain expert needs approximately 2.256 days to manually locate each feature. Therefore, it would take 22.56 days to locate the 10 features.

This number does not consider the experience of the engineers. A newly hired engineer can take up to a year to fully master the architecture of the models. Also, even if the engineers had experience, the domain experts usually do not know all models completely. The models can be created by several different domain experts. Furthermore, they can forget the details of a model after developing it. The domain experts can forget models that belong to trains manufactured over two decades. Time because of the learning effect, no existing documenta-

tion, or locating several features simultaneously are not accounted here, but these could also be source of errors that take time to fix.

Thus, feature location in real-world models is not a trivial task. The 10 feature descriptions used in this work correspond to tramway models. However, the need to locate features is also present in other CAF models of similar complexity such as subway models, or in more complex models such as suburban and high-speed models.

2.2 Feature location in models based on machine learning

Several different model fragments can be generated from the elements of a model. Therefore, in order to know which model fragment is the best realization of a feature, each model fragment has to be evaluated with regard to the feature description. Different techniques can be used to evaluate the model fragments regarding the feature description. However, machine learning techniques take advantage of the model fragments and feature descriptions that have been evaluated previously. For example, an engineer of our industrial partner identified the model fragment shown in Fig. 1 as the best realization of the "car door state signaling" feature. Therefore, a machine learning technique can use this model fragment and the description of that feature to learn rules such as *"if the feature description contains the term door, the model fragment has to contain at least one door of the model"*. These learned rules can be applied to evaluate other model fragments for other features.

However, to understand the model fragments and the feature descriptions, the machine learning techniques require the inputs to be in a specific format. Most of the machine learning techniques are designed to process feature vectors as inputs [10]. Feature vectors are known to be the ordered enumeration of characteristics that describe the object being observed [16]. In feature location in models, the object being observed is a model fragment regarding a feature description. Therefore, to understand the model fragment in Fig. 1, the machine learning technique requires that the model fragment and the feature description be encoded into a feature vector.

Each feature vector consists of feature-value pairs. However, the concept of feature could be confused in this specific context because it has two meanings. On the one hand, in feature location, a feature is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system [51,44]. On the other hand, in machine learning, a feature is an individually measurable characteristic of the object being observed [16]. To avoid misunderstandings, in this article, the term feature regarding feature vectors is replaced by the term characteristic. Therefore, each feature vector consists of characteristic-value pairs, where each characteristic helps to describe the object being observed. For example, taking into account the model fragment in Fig. 1 and the "car door state signaling" feature, a characteristic of the feature vector could be the number of doors in the model fragment. If the value of this characteristic is 0, the model fragment is not the best realization of the feature because the feature needs at least one door. In contrast, if the value of this characteristic is 1, the model fragment could be the best realization of the feature. Nevertheless, we would need other characteristics to determine if this model fragment is the best realization of the feature or other model fragments are better.

In summary, the encoding enables the feature descriptions and model fragments to be represented as feature vectors, which can be understood and manipulated by machine learning techniques. In other words, the feature descriptions and model fragments, which could not be used by machine learning techniques, are encoded into feature vectors to be exploited by these techniques. The feature vectors are used to train a classifier, and the classifier is used to determine if a certain model fragment is a better realization of a feature than other fragments. Therefore, the first step in locating features in models using machine learning techniques is the encoding.

3 Encodings

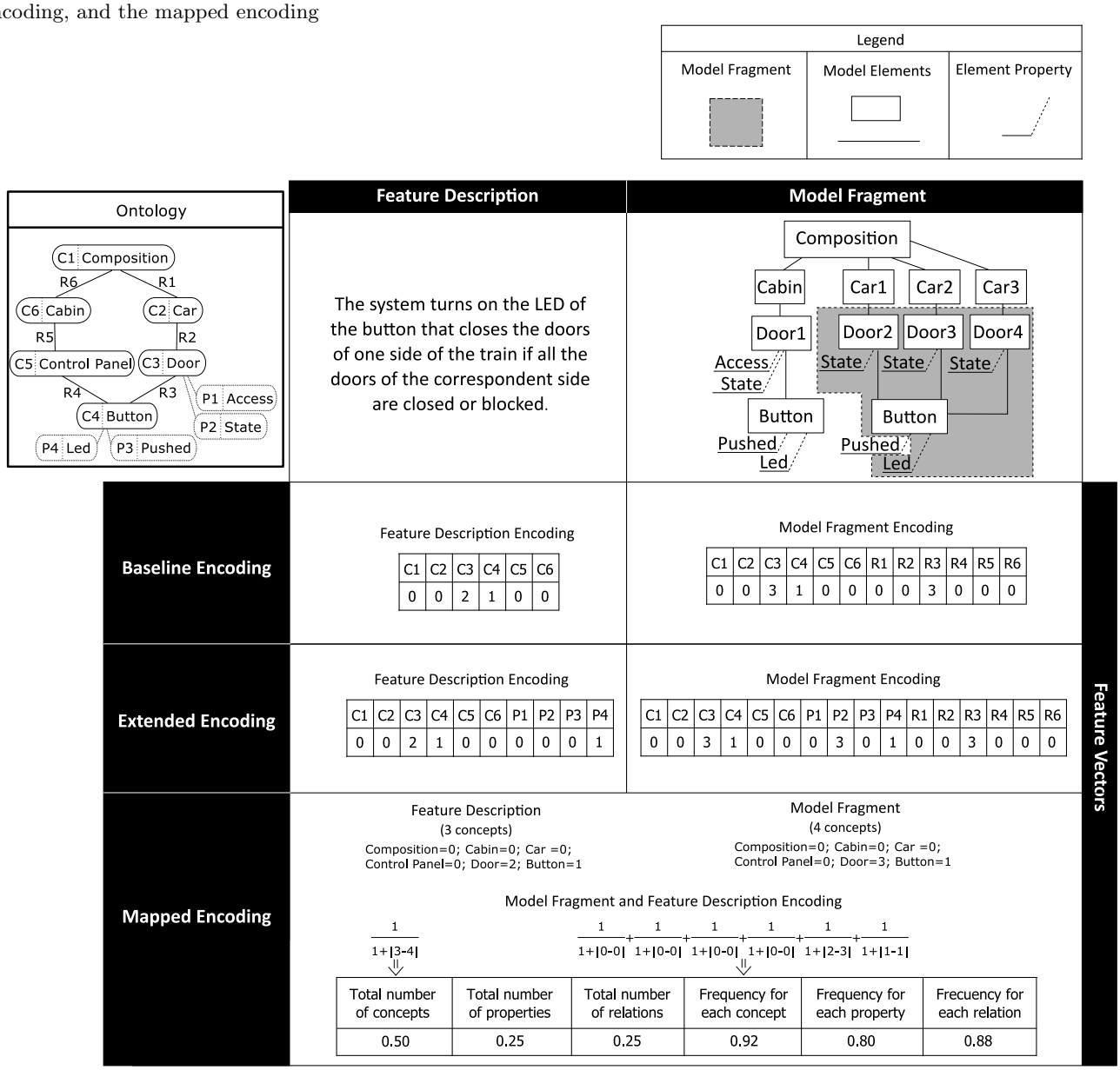
In this section, we present three ontology-based encodings to turn the feature descriptions and model fragments into feature vectors. It is noteworthy that all the encodings use a combination of the natural language processing techniques defined in [48] to homogenize the terms in feature descriptions and model fragments. The first encoding was presented in our previous work [60] and is used as the baseline here. The other two encodings have been designed taking into account the whole domain ontology and the characteristics used in benchmark datasets, respectively.

3.1 The Source Encoding

This encoding is based on the main concepts and relations of a domain ontology. In this encoding, the feature descriptions are encoded taking into account the concepts of the ontology. Specifically, the feature vector that is created from a feature description contains as many characteristics as concepts in the ontology. The value of each characteristic is computed as the frequency of a concept in the feature description.

The first row of Fig. 2 shows an example of an ontology, feature description, and model fragment. The second row shows the source encoding for the feature description and the model fragment. The domain ontology contains six concepts, which are tagged with the letter C and a number. Since there are six concepts, the feature description is encoded using six characteristic-value

Fig. 2 Example of the encoding of a feature description and a model fragment based on the source encoding, the extended encoding, and the mapped encoding



pairs. The characteristics correspond to the tags of the concepts (e.g., *Door* is mapped as *C3*), and their values correspond to the frequency of the concept in the feature description (e.g., *C3* has a value of 2 because the word *Door* appears twice in the feature description).

In contrast, the model fragments are encoded taking into account not only the concepts of the ontology but also its relations, which are tagged with the letter *R* and a number. Specifically, the feature vector created from a model fragment contains as many characteristics as concepts and relations in the ontology. The value of each characteristic is computed as the frequency of a concept or relation in the model fragment.

Taking into account the second row of Fig. 2, the feature vector is encoded using not only the six concepts of the domain ontology but also the six relations of this ontology. Therefore, its feature vector contains twelve characteristic-value pairs. In this example, the characteristic *C3* has a value of 3 because the model fragment contains three elements of *Door* type and the characteristic R5 has a value of 0 because the model fragment does not contain any relation of *Cabin-Control Panel* type.

3.2 The Extended Encoding

This second encoding is based on the main concepts, properties, and relations of a domain ontology. In other words, this encoding is an extension of the source encoding taking into account the whole ontology. The source encoding provides a lower number of characteristics,

which usually leads to better learning performance (e.g., higher learning accuracy for classification), lower computational cost, and better classifier interpretability [69, 94]. However, taking into account only the concepts and relations of the ontology may not allow the differentiation between the elements of the same type in the model fragments. For example, in the model fragment of Fig. 2, there are four elements of *Door* type so a reasonable question could be why the *Door1* element is not part of the model fragment. The difference between the *Door1* element and the other elements of *Door* type can lie in their properties. For this reason, this extended encoding is proposed to include more details in feature vectors about both the feature descriptions and the model fragments.

The concepts and relations of the ontology are the same as in the previous encoding, but, in this case, the feature vectors generated also contain the characteristic-value pairs for the properties of the ontology. The value of each characteristic is computed as the frequency of the property in the feature description or the model fragment, respectively.

The third row of Fig. 2 shows an example of this encoding, where a feature description and a model fragment are encoded based on the extended ontology. This encoding is based on six concepts, four properties, and six relations. The properties are tagged using the letter P and a number. For example, the characteristic *P4* has a value of 1 in the encoded feature description because the word *LED* appears once in the feature description. In the encoded model fragment, *P4* has a value of 1 because it contains one element of type *Button* with the *LED* property.

3.3 The Mapped Encoding

While the source encoding provides a starting point to bridge machine learning and models, the extended encoding focuses on providing more details than the source encoding. However, the more concepts, properties, and relations there are in an ontology, the more characteristic-value pairs there are in a feature vector. Although this favors the level of detail, a high number of characteristics may lead to worse results in most cases [69, 94]. To reduce this threat, the mapped encoding focuses on reducing the number of characteristics taking into account the characteristics used in benchmark datasets like [76].

These characteristics are widely used in the research community to encode raw data such as text and url. The mapped encoding adapts the *sum of term frequency* and the *mean of term frequency* characteristics taking into account the frequency of the concepts, the properties, and the relations instead of the frequency of the terms. Furthermore, instead of a single text, the mapped encoding must encode two artifacts, a feature description and a model fragment. To do this, the mapped encoding takes into account the difference between the frequencies in the feature description and in the model fragment.

The *sum of term frequency* characteristic was adapted using the following equation:

$$\frac{1}{1 + |\text{Number of I in FD} - \text{Number of I in MF}|} \quad \begin{array}{r} \text{where I are concepts, properties, or relations;} \\ FD \text{ is a feature description;} \\ MF \text{ is a model fragment} \end{array} \tag{1}$$

Three characteristics of the mapped encoding are based on this equation. The first one compares the total number of concepts in the feature description to the total number of concepts in the model fragment. The second one compares the total number of properties in the feature description to the total number of properties in the model fragment. The third one compares the total number of relations in the feature description to the total number of relations in the model fragment.

The fourth row of Fig. 2 shows an example of the characteristics in the mapped encoding. The first three characteristics compare the feature description and the model fragment regarding the total number of concepts, the total number of properties, and the total number of relations. In this example, the *Door* concept appears twice and the *Button* concept appears once in the feature description, so the number of concepts in the feature description is equal to 3. The *Door* concept appears three times in the model fragment (i.e., *Door1*, *Door2*, and *Door3*) and the *Button* concept appears once, so the number of concepts in the model fragment is equal to 4. Through Equation (1), the first characteristic is computed as $1/1 - |3 - 4|$ which is equal to 0.5. Similarly, the second characteristic takes into account the properties instead of the concepts and the third characteristic takes into account the relations.

The *mean of term frequency* characteristic was adapted using the following equation:

$$\frac{\sum_{i=1}^{n} \frac{1}{1 + \left|\begin{array}{l}\text{frequency of} \\ I_i \text{ in FD}\end{array} - \begin{array}{l}\text{frequency of} \\ I_i \text{ in MF}\end{array}\right|}}{n} \quad \begin{array}{r} \text{where I are concepts, properties, or relations;} \\ FD \text{ is a feature description;} \\ MF \text{ is a model fragment} \end{array} \tag{2}$$

Three characteristics of the mapped encoding are based on this equation. The first one compares the frequency of each concept in the feature description to the frequency of the corresponding concept in the model fragment. The second one compares the frequency of each property in the feature description to the frequency

of the corresponding property in the model fragment. The third one compares the frequency of each relation in the feature description to the frequency of the corresponding relation in the model fragment.

In the fourth row of Fig. 2, the last three characteristics of the mapped encoding compare the feature description and the model fragment regarding the frequency of each concept, property, and relation. In this example, the *Composition* concept does not appear in the feature description or in the model fragment, so the frequency of the *Composition* concept is computed as $1/(1+|0-0|)$ which is equal to 1. Similarly, the *Cabin*, *Car*, and *Control Panel* concepts do not appear in the feature description or in the model fragment, so the frequency of these concepts is equal to 1. The *Door* concept appears twice in the feature description and three times in the model fragment (i.e., *Door1*, *Door2*, and *Door3*), so the frequency of the *Door* concept is computed as $1/(1+|2-3|)$ which is equal to 0.5. The *Button* concept appears once in the feature description and once in the model fragment, so the frequency of the *Button* concept is computed as $1/(1+|1-1|)$ which is equal to 1. Following Equation (2), the mean of these values is equal to 0.92, which is the value of the fourth characteristic in the mapped encoding. Similarly, the fifth characteristic takes into account the properties instead of the concepts and the sixth characteristic takes into account the relations.

Therefore, the mapped encoding is composed of six characteristics. Three characteristics respectively compare the total number of concepts, properties, and relations in feature descriptions and model fragments. Three characteristics respectively compare the frequency of each concept, property, or relation in the feature description and the model fragment.

4 Evaluation

The goal of this evaluation is to provide answers to the following research questions for feature location in models.

RQ1: Which of the three encodings yields the best classification performance when combined with the RankBoost classification technique?

RQ2: Taking into account the three encodings, which machine learning technique obtains the best results?

RQ3: Taking into account the encoding and the machine learning technique that obtained the best results in the previous questions, can a domain-independent encoding provide better results than an encoding designed by a domain expert exploiting both human experience and domain knowledge?

RQ4: Is a machine learning-based approach better than more traditional approaches (linguistic rule-based and information-retrieval)?

This section presents the evaluation that was performed to answer the RQs. It includes the following: the experimental setup, a description of the case studies where we applied the evaluation, the FLiM-ML approach, the machine learning techniques, the implementation details, and the obtained results.

4.1 Experimental Setup

To provide answers to the RQs, the experiment consisted of configuring the FLiM-ML approach using a specific encoding and a machine learning technique. Then, we compared the quality of the results in terms of precision, recall, F-measure, and Matthews Correlation Coefficient (MCC) [65].

Specifically, the experiment applied the FLiM-ML approach with each possible combination between four encodings and three machine learning techniques. The first three encodings were the ones proposed in this work: the source encoding **(Sou)**, the extended encoding **(Ext)**, and the mapped encoding **(Map)**. The fourth encoding was designed by domain experts who were not involved in the research: the human encoding **(Hum)**. Each human encoding was specifically designed for a case study, so a human encoding of a case study is different from the encodings of other case studies.

The first machine learning, RankBoost **(Rank)**, belongs to the family of machine learning techniques that automatically address ranking tasks. RankBoost was previously selected taking into account its efficiency and effectiveness to test the source encoding [60]. The last two machine learning techniques are deep learning techniques: Feedforward Neural Network **(FNN)** and Recurrent Neural Network **(RNN)**. They have been successfully applied to retrieval problems in recent works [35].

The FLiM-ML approach evaluates the different test cases by combining an encoding with a machine learning technique. For example, SouRank evaluates all the test cases through the FLiM-ML approach using the source encoding and the RankBoost technique. All the possible combinations of the four encodings and the three machine learning techniques were used by the FLiM-ML approach to test all the test cases.

Fig. 3 shows an overview of the process that was followed to evaluate the approach using each combination of an encoding and a machine learning technique. Fig. 3 (left) shows the inputs, which are extracted from the documentation provided by our industrial partners: knowledge base, ontology, feature descriptions, product models, and approved solutions. Each test case is comprised of a feature description, a set of model fragments, the ontology, and the knowledge base. Then, each test case was run 30 times. As suggested by [4], given the stochastic nature of the FLiM-ML approach, several repetitions are needed to obtain reliable results. Finally, the

results of the test cases were evaluated and compared to the oracle. The oracle is composed of the approved solutions, which are the model fragments that correctly realize a feature description of the case study.

For each execution of the FLiM-ML approach, the approach generates a ranking of model fragments. Each model fragment realizes the feature to a greater or lesser extent. Then, we take the best model fragment of the ranking and compare it against the oracle, which is the ground truth. In our case, each solution that is output by the approach is a model fragment composed of a subset of the model elements that are part of the product model. Since the granularity is at the level of model elements, the presence or absence of each model element is considered to be a classification. Therefore, for each test case, a confusion matrix distinguishing between the predicted values and the real values is calculated. Then, some performance measurements are calculated from the values in the confusion matrix. Specifically, we create a report that includes four performance measurements (precision, recall, F-measure, and MCC) for each test case that is tested through the FLiM-ML approach.

Precision values can range between 0% (i.e., no single model element from the solution is present in the oracle) and 100% (i.e., all the model elements from the solution are present in the oracle). A value of 100% precision and 100% recall implies that both the solution and the feature realization from the oracle are the same. Recall values can range between 0% (i.e., no single model element from the realization of the feature description obtained from the oracle is present in the model fragment of the solution) and 100% (i.e., all the model elements from the oracle are present in the solution). MCC values can range between -1 (i.e., there is no correlation between the prediction and the solution) to 1 (i.e., the prediction is perfect). Moreover, a MCC value of 0 corresponds to a random prediction.

4.2 Case Studies

The case studies where we applied our approach were provided by CAF, which is a worldwide provider of railway solutions, and BSH, which is one of the largest manufacturers of home appliances in Europe. CAF trains can be found all over the world and in different forms (regular trains, subway, light rail, monorail, etc.). A train unit is furnished with multiple pieces of equipment in its vehicles and cabins. These pieces of equipment are often designed and manufactured by different providers, and their target is to carry out specific tasks for the train. Some examples of these devices are: the traction equipment, the compressors that feed the brakes, the pantograph that harvests power from the overhead wires, and the circuit breaker that isolates or connects the electrical circuits of the train. The control software of the train unit is in charge of making all the equipments cooperate to achieve the train functionality while guaranteeing compliance with the specific regulations of each country. The following video illustrates the CAF models: `http://youtube.com/watch?v=Ypcl2evEQB8`

BSH is a leading manufacturer of home appliances in Europe. Its induction division has been producing induction hobs (sold under the brands of Bosch and Siemens) for the last 15 years. The firmware that controls the induction hobs is specified by means of a domain-specific language (IHDSL[4]). The different configurations of the induction hobs are managed following a model-based software product line (SPL) approach that uses common variability language [39] to configure their models. The firmware of their products is generated from the IHDSL models.

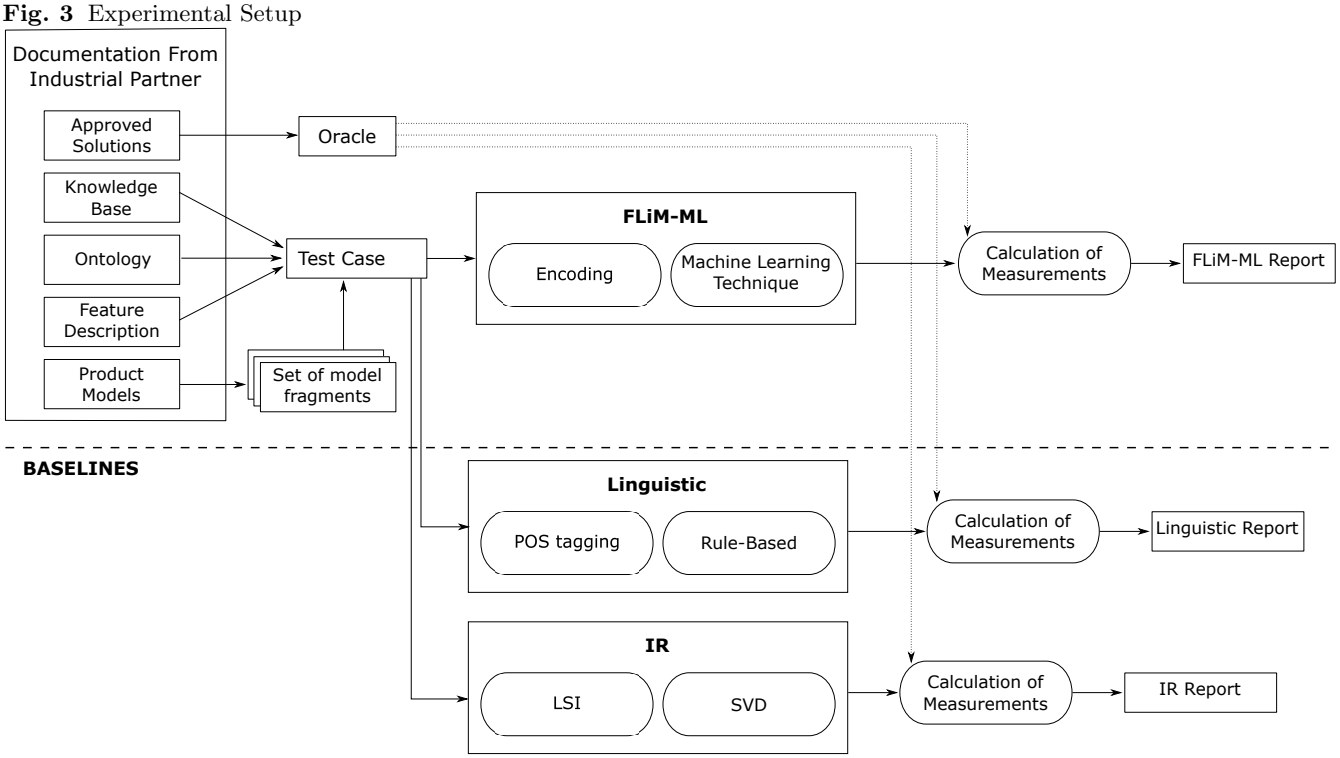
Each one of the companies provides the necessary documentation to perform the experiment described in Fig. 3. The documentation provided by CAF was used to evaluate the encodings and the machine learning techniques reporting the results for CAF. The documentation provided by BSH was used to evaluate the same encodings and the machine learning techniques reporting the results for BSH. The documentation of both case studies was managed independently; when the CAF documentation was used for training, the BSH documentation was not used for testing, or vice versa.

The documentation provided for both cases studies was organized into a set of test cases and the oracle. Each test case is composed of a feature description, a set of model fragments, a knowledge base, and an ontology. A detailed description of each is provided below:

- The **feature descriptions** describe a feature of a train control system or an induction hob using natural language. All the feature descriptions are located in all the model fragment sets of the case study.
- The **sets of model fragments** are composed of model fragments selected from product models. Specifically, our industrial partners provided several product models, whose size is so big that would take months of work to evaluate all the model fragments for each product model. For example, a CAF model has about 650 elements. If we take into account all the possible combinations of these elements (i.e., $C(n,k)$ combinations of size k from a set of n distinct elements), we will obtain 650 model fragments with a single element (i.e., $C(650,1)$), 21,0925 model fragments with two elements (i.e., $C(650,2)$), and we will continue until to find a model fragment with all the model elements (i.e., $C(650,650)$). In total, it would be possible to find $4672x10^{196}$ different model fragments in the CAF model.
  Therefore, instead of evaluating all the model fragments of a product model, we randomly selected the model fragments for our evaluation and grouped them into representative sets. These sets are related

[4] Learn more of IHDSL at: https://youtu.be/nS2sybEv6j0

Fig. 3 Experimental Setup



to the different location challenges that are usually tackled by our industrial partners. Our evaluation considered nine different location challenges, which are associated in [6] to three different measurements: density, multiplicity, and dispersion.

Density measures the percentage of model elements that are contained by the model fragment. This measurement is associated with three location challenges. The first one is the search for a small model fragment in a large product model. The second one is the search for a large model fragment in a small product model. The third one is the search for a model fragment in a product model without considering the size of the model fragment to be located. These location challenges were evaluated through a set of model fragments with small density values (between 0% and 50%), a set of model fragments with large density values (between 50% and 100%), and a set of model fragments with all kinds of density values, respectively.

Multiplicity measures the number of times the model fragment appears in the product model. This measurement is also associated with three location challenges. The first one is the search for a model fragment that appears only once in a product model. The second one is the search for a model fragment that appears several times in a product model. The third one is the search for a model fragment in a product model, without considering the number of times the model fragment appears. These location challenges were evaluated through a set of model fragments with multiplicity values equal to 1, a set of model fragments with multiplicity values greater than 1, and a set of model fragments with all kinds of multiplicity values, respectively.

Finally, dispersion measures the ratio of connected elements in the model fragment. This measurement is associated with the last three location challenges. The first one is the search for a model fragment whose elements are connected with each other. The second one is the search for a model fragment whose elements are not connected. The third one is the search for a model fragment in a product model without considering the connections among the elements in the model fragment. These location challenges were evaluated through a set of model fragments with small dispersion values (between 0 and 0.5), a set of model fragments with large dispersion values (between 0.5 and 1), and a set of model fragments with all kinds of dispersion values, respectively.

Therefore, for each product model, the selected model fragments are grouped into nine sets of model fragments to evaluate the different location challenges.

– The **knowledge base** is a collection of retrieved model fragments that are related to feature descriptions and scores. Companies that have been developing software systems for a long time have gathered a large amount of knowledge and experience. In fact, engineers and modelers usually collect the model

fragments that they manually retrieve to maintain the software systems. Moreover, the quality of the retrieved model fragment depends on the experience of the engineer, the complexity of the description, the complexity of the product model, and the time required. Sometimes, manual searches can be unsuccessful or incomplete. For this reason, a domain expert usually assigns scores to indicate how good the model fragments are. Scores are numerical values that are greater than 0 and indicate the degree of similarity between the model fragment and feature description. Then, the model fragments are stored in a knowledge base with their scores and the correspondent feature descriptions.

– The **ontology** is composed of the main concepts, properties, and relations of a domain; therefore, if a relevant concept, property, or relation is not present in the ontology, it will not be considered by the encodings and the results may not be as good. Also, if the ontology contains unnecessary concepts, properties, or relations, the number of characteristics in the encodings would be greater and a large number of features in machine learning leads to overfitting. Therefore, it is important to keep in mind both the completeness and the size of the ontology.

In addition to the test cases, our partners also provided us with the approved solution to have an oracle. In other words, the oracle is composed of the model fragments that are the correct solutions for the feature descriptions. After evaluating each test case, the result of the FLiM-ML approach is compared with the correspondent correct solution that is available in the oracle.

Table 1 summarizes the case studies for both domains (the railway domain and the induction hob domain).

The **CAF case study** includes 1800 test cases, which are composed of a feature description, a set of model fragments, a knowledge base, and an ontology. The case study contains 10 different feature descriptions, each one of which has about 25 words and describes a feature of a train control system. Specifically, there are 180 different sets of model fragments, which come from tackling the 9 different location challenges in 20 different product models. All the feature descriptions are located in all the sets of the model fragment.

Each test case also contains a knowledge base, which has 1339 samples. Each of these samples contains a feature description with about 25 words, a model fragment with about 15 elements, and a score between 0 and 4. The knowledge base is balanced to have a similar number of samples with low and high scores. The test cases contain a domain ontology, which has a total of 103 elements that represent concepts, properties, and relations.

In addition to the test cases, CAF provided an oracle with 10 model fragments. These model fragments are the correct solution for the 10 feature descriptions in the test cases.

The **BSH case study** includes 108 test cases, which are composed of a feature description, a set of model fragments, a knowledge base, and an ontology. The case study contains 6 different feature descriptions, each one of which has about 10 words and describes a feature of an induction hob. Specifically, there are 180 different sets of model fragments, which come from tackling the 9 different location challenges in 2 different product models. All the feature descriptions are located in all the sets of model fragments.

Each test case also contains a knowledge base, which has 758 samples. Each of these samples contains a feature description with about 10 words, a model fragment with about 8 elements, and a score between 0 and 4. The test cases contain a domain ontology, which has a total of 13 elements that represent concepts, properties, and relations.

In addition to the test cases, BSH provided an oracle with 6 model fragments. These model fragments are the correct solution for the 6 feature descriptions in the test cases.

4.2.1 The Human Encodings

The human encodings were also provided by our industrial partners, one for the railway domain and one for the induction hob domain. The domain experts who designed the human encodings knew that these encodings would be used to perform feature location in their models. Therefore, the encodings had to tackle not only the model fragments but also the feature descriptions. The experts were also informed about the machine learning techniques to be used for the feature location, and we even suggested that they could propose more than one encoding.

Each encoding was designed by a domain expert who was not involved in the research. In order to mitigate the dependence on a single domain expert, a second expert who also was not involved in the research extended the encoding. However, there were no significant differences between the results obtained using the initial encodings and the results obtained using the extended ones. Specifically, the results described in this work correspond to the extended encodings, which are a bit better (about 1% better) than those obtained from the initial encodings.

The domain experts took about two hours to identify the characteristics for each encoding. Some of the identified characteristics were very similar or even equal to the ones proposed in our encodings. For example, in the induction hob domain, one of the identified characteristics was the number of inductors. This characteristic is also considered in the source encoding and in the extended encoding. However, none of the domain experts identified or proposed characteristics that were similar to the ones in the mapped encoding. Once the characteristics

Table 1 Summary of the two case studies provided by our industrial partners: CAF and BSH

	CAF Case Study	**BSH Case Study**
Test Cases	1800 Test Cases	108 Test Cases
Feature Descriptions	10 Feature Descriptions with about 25 words	6 Feature Descriptions with about 10 words
Sets of Model Fragments	180 Sets of Model Fragments from 20 product models	18 Sets of Model Fragments from 2 product models
Knowledge Base	1339 samples with: – A Feature Description (about 25 words) – A Model Fragment (about 15 elements) – A score (between 0 and 4)	758 samples with: – A Feature Description (about 10 words) – A Model Fragment (about 8 elements) – A score (between 0 and 4)
Ontology	103 elements between concepts, properties, and relations	13 elements between concepts, properties, and relations
Oracle	10 model fragments	6 model fragments

were identified, the feature descriptions and the model fragments were encoded automatically as in the rest of the encodings. No one manually counted the occurrences of a term or an element.

In the case of the railway domain, the human encoding contains a total of twelve characteristics. These characteristics were based on the category of the elements (i.e., equipment, property, order). To encode the feature descriptions, the human encoding took into account five characteristics: the number of the terms related to equipments, the presence or absence of terms related to rules, the number of the terms related to properties, the number of the terms related to orders, and the presence or absence of terms related to conditions. To encode the model fragments, the human encoding took into account seven characteristics: the number of equipment-type elements, the presence or absence of rule-type elements, the number of property-type elements, the number of order-type elements, the presence or absence of condition-type elements, the number of trigger-type elements, and the number of action-type elements.

Note that the human encoding for the railway domain has a different perspective than the rest of the encodings. While the others use the main concepts of the domain (i.e., pantograph, circuit breaker, door, cabin), this human encoding uses the category of the elements (i.e., equipment, property, order). This encoding also contains some characteristics based on the number of terms or elements and other characteristics based on Boolean values to indicate the presence or absence of specific terms or elements.

In the case of the induction hob domain, the human encoding contains a total of six characteristics. To encode the feature descriptions, this encoding took into account three characteristics: the number of the term *inductor*, the number of the term *inverter*, and the number of the term *power manager*. To encode the model fragments, this encoding also considered three characteristics: the number of Inductor-type elements, the number of Inverter-type elements, and the number of Power manager-type elements.

4.3 FLiM-ML Approach

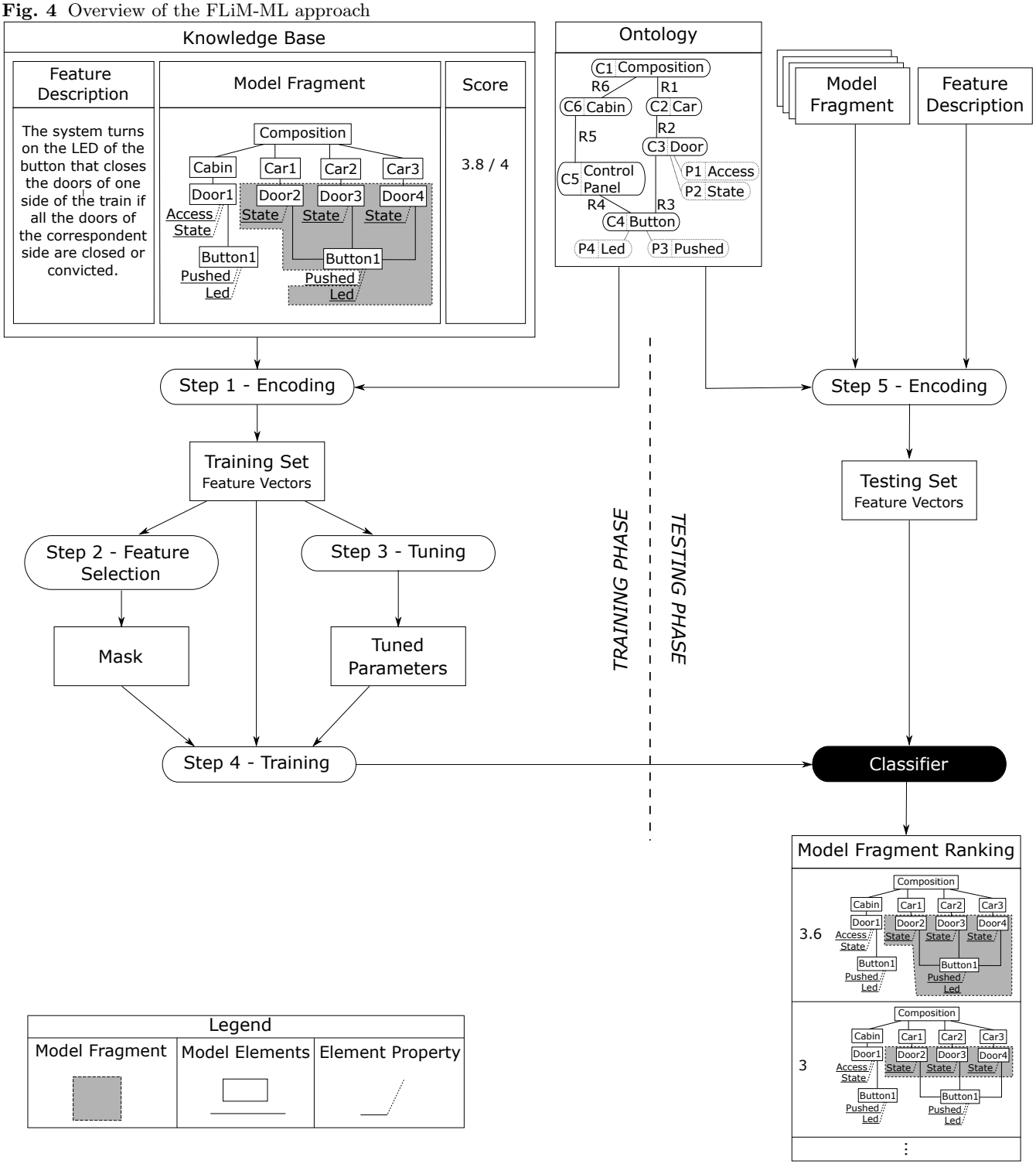
The FLiM-ML approach was presented in [58,59] and was used to evaluate the source encoding in [60]. In this work, the FLiM-ML approach is used to evaluate the test cases using different encodings and machine learning techniques. Fig. 4 shows an overview of the FLiM-ML approach, whose objective is to provide a ranking of model fragments. The top model fragment in the ranking is the best realization found for a feature description. To do this, the approach has two phases: training and testing.

In the training phase, a classifier is trained to learn how well each model fragment realizes a specific feature description. To do this, the input consists of a knowledge base and an ontology. The knowledge base is composed of samples, each of which relates a feature description and a model fragment according to a score. The ontology is composed of concepts, properties, and relations of a domain.

The training phase consists of four steps:

1. Encoding: An ontology is used to encode the samples of the knowledge base into feature vectors. Since both feature descriptions and model fragments are based on natural language, the terms used in the ontology do not always align with the terms in the feature description and with the terms in the model fragments. For this reason, before encoding, the feature descriptions and the model fragments are processed by a combination of natural language processing techniques defined in [48], which consists of tokenizing, lowercasing, removal of duplicate keywords,

Fig. 4 Overview of the FLiM-ML approach



syntactical analysis, lemmatization, and stopword removal. Then, all the samples of the knowledge base are encoded into feature vectors and these feature vectors are the training dataset.

2. Feature selection: A mask is applied to select only the most relevant characteristics in the feature vectors, which reduces the time cost and the redundant information [9,13]. In a mask, each characteristic of the feature vectors has a binary value (discard or select). For example, the characteristic *C1* may be 0 in a mask and 1 in another one. In the first mask, the characteristic *C1* would be discarded, so the feature vectors would be simplified removing this characteristic. In the second mask, the characteristic *C1* would be selected, so all the feature vectors must have this characteristic. A set of different masks is generated,

evolved, and tested by means of an evolutionary algorithm as in [60]. As a result, the mask which obtains the best results is applied to select the characteristics.

3. Tuning: This determines what parameters must be used to obtain the best performance of the machine learning technique. The parameter tuning in the FLiM-ML approach is automatically performed based on the machine learning technique used.
4. Training with a machine learning technique: The training set is used to train the classifier, which learns a rule-set through the comparison of the feature vectors of the training set [82]. However, before using this classifier in the testing phase, it is worth analyzing the performance of the classifier through cross-validation. Cross-validation is a statistical method of evaluating and comparing machine learning techniques by dividing data into two segments: one is used to train a classifier, and the other is used to validate the classifier [78]. Moreover, to reduce variability, multiple rounds of cross-validation are performed using different partitions, and the results are averaged over the rounds [83].
   The results of the cross-validation provide the performance of the classifier. If this performance is not considered suitable, it is necessary to perform another training iteration. In this iteration, some artifacts of the training phase (e.g., the encoding, the ontology, the knowledge base, or the machine learning technique) must be modified in order to improve the classifier. Otherwise, if the performance is considered suitable by the engineers, the classifier obtains the go-ahead, so the classifier trained with the whole knowledge base is used in the testing phase. Once the classifier has been generated, the training phase is not repeated again. The same classifier is used to evaluate all the test cases in the testing phase. Therefore, the classifier is considered both as an artifact (output of Step 4 in the training phase) and a step (responsible for testing the test cases in the testing phase). For this reason, Fig. 4 shows the classifier in a black, rounded rectangle to point out its double meaning.

In the testing phase, the classifier is used to rank the set of model fragments according to a feature description described in natural language. To do this, the input consists of the set of model fragments, the feature description, and an ontology. Each model fragment realizes the feature description to a greater or lesser extent.

The testing phase consists of two steps:

5. Encoding: An ontology is used to encode each model fragment and the feature description. To be fair, both the characteristics of the encoding and the ontology must be the same for the training phase and the testing phase. As a result, each model fragment and the feature description are encoded as a feature vector in the testing set.
6. Classifier: The classifier is responsible for automatically assigning a score to each feature vector in the testing dataset. Specifically, the classifier contains the set of rules, which are identified or learned during the training process. To assign the scores, these rules are based on different mathematical procedures that differ for each machine learning technique. Therefore, each feature vector in the testing dataset is tested by the classifier applying the rule-set. As a result, a score is assigned to each feature vector. Since each feature vector is the representation of a model fragment, we can relate each model fragment to the score assigned by the classifier to its correspondent feature vector. The higher the score, the closer the model fragment is to the feature description. Therefore, this model fragment would be a better realization of the feature description than any other model fragment with a lower score. Taking into account the scores, the model fragments can be ordered in a ranking where the top positions are occupied by the model fragments that have the highest relevance to the feature description. This ranking is the final result of the FLiM-ML approach.

4.4 Machine Learning techniques

In the evaluation, we used three different machine learning techniques to see how the machine learning technique affects the quality of the result. Since our previous work focused on RankBoost, using it in this work seemed to be the natural step for our ongoing research. Nevertheless, the recent success of deep learning techniques in search problems [35,19,20] has encouraged us to also evaluate those techniques in comparison with RankBoost.

4.4.1 RankBoost

RankBoost [31] belongs to the family of Learning to Rank algorithms and is well known for its efficiency and effectiveness in different domains [14,15]. RankBoost can benefit from a small knowledge base together with a small number of characteristics in the encoding to reduce the overfitting problem [95,88].

To apply RankBoost, the tuning was performed as in [45]. First, a grid search is built to determine the values of the parameters. Then, the FLiM-ML approach uniformly samples each of the parameters in their range and evaluates all the combinations of the sampled values. The RankBoost algorithm receives as input the number of iterations that the algorithm will perform and the threshold corresponding to the number of candidates to be considered in the weak rankers [67]. Moreover, the metric to be optimized on the training set is often reported.

Table 2 shows the values obtained from feature selection and the specific values tuned for each encoding-technique pair. Specifically, the grid search was used to determine the number of iterations and the threshold value. In contrast, the selected metric was the default metric of the RankLib library [21] that was used to implement RankBoost. The values considered for the number of iterations were in the range [100,500]. The values considered for the threshold were in the range [2,10].

4.4.2 Feedforward Neural Network

Feedforward Neural Networks (FNNs) represent a traditional neural network structure and lay the foundation for many other structures [40]. Data flow always moves one in direction, from input layer to hidden layer, then to output layer; it never goes backward. A FNN can have more than one hidden layer. However, it has been proven that FNNs with monotonically increasing differentiable functions can approximate any continuous function with one layer provided that the hidden layer has enough hidden neurons [43]. For this reason, the network architecture of the FNN implemented for the evaluation is a dense layer that is followed by the final softmax layer.

For each encoding and case study, we performed a hyperparameter optimization based on the random search optimization provided by the Deep Learning for Java library. The Deep Learning for Java library also provided a grid search for optimization. However, the number of parameters for tuning in neural networks is higher than in RankBoost. While RankBoost has two parameters for tuning, Neural Networks have at least four parameters for tuning. Therefore, to improve the efficiency of the tuning as suggested in [8], we decided to use random search to tune the neural networks.

Table 3 shows the values obtained from feature selection and the specific values tuned for each encoding and case study. The parameters tuned were the initial learning rate, the weight initialization, the layer size, and the activation function. The values considered for the initial learning rate were in the range [0.0001, 0.1]. The values considered for the weight initialization were normal [46], Glorot normal [33], and sigmoid uniform, which is a version of Glorot uniform for sigmoid activation functions [33]. The values considered for the layer size were in the range [128, 256]. Finally, the values considered for the activation function were the logistic sigmoid function, hardsigmoid function, tanh function, hardtanh function, Rectified Linear Unit (ReLU) function, and Randomized Rectified Linear Unit (RReLU). More details about the parameters, such as the default values or the formulas, are available in the library guide [26].

4.4.3 Recurrent Neural Network

Since the number of parameters in a fully connected FNN can grow extremely large as the width and depth of the network increase, researchers have proposed other neural network structures, such as Recurrent Neural Networks (RNNs). While FNNs have no feedback connections to previous layers, RNNs have these feedback connections to model the temporal characteristics of the problem being learned [27]. RNNs are usually used for sequential data, so this technique may not be the most appropriate for the models of our case study. However, the recent success of RNNs in search problems [35] has encouraged us to evaluate this technique in comparison with other machine learning techniques that, a priori, are more appropriate for our case study (e.g., RankBoost and FNNs). Specifically, the RNN implemented for the evaluation contains a Long Short Term Memory (LSTM) layer followed by the final softmax layer.

Table 4 shows the values obtained from feature selection and the specific values tuned for each encoding and case study. The initial learning rate, the weight initialization, the layer size, and the activation function were the parameters tuned through the random search optimization, which is provided by the Deep Learning for Java library. The initial learning rate was assigned taking into account values in the range [0.0001, 0.1]. The values considered for the weight initialization were normal [46], Glorot normal [33], and sigmoid uniform [33]. The values considered for the layer size were in the range [128, 256]. Finally, the values considered for the activation function were the logistic sigmoid function, hardsigmoid function, tanh function, hardtanh function, Rectified Linear Unit (ReLU) function, and Randomized Rectified Linear Unit (RReLU). More details about the parameters, such as the default values or the formulas, are available in the library guide [26].

4.5 Baselines

In addition to evaluating the encodings taking into account three different machine learning techniques, we compared the results obtained with the traditional approaches for feature location (see Fig. 3). We compared the best results of the FLiM-ML approach with the results of a Linguistic Rule-based (Linguistic) approach [84] and with the results of an Information Retrieval (IR) approach based on Latent Semantic Indexing [22, 55]. These two approaches are used successfully not only in feature location but also in other software tasks that require the location of model fragments. In fact, they are the best approaches for requirements traceability in models according to the classification in Winkler et al. [90].

Table 2 RankBoost setup for the FLiM-ML approach depending on the encoding and the case study

		Feature Selection		Hyperparameters		
		Total number of characteristics	Number of selected characteristics	Number of iterations	Threshold	Metric
CAF	SouRank	65	60	100	8	ERR10
	ExtRank	103	88	100	10	ERR10
	MapRank	6	6	100	8	ERR10
	HumRank	12	12	100	10	ERR10
BSH	SouRank	13	11	100	4	ERR10
	ExtRank	21	19	100	10	ERR10
	MapRank	6	6	100	8	ERR10
	HumRank	6	4	100	10	ERR10

Table 3 FNN setup for the FLiM-ML approach depending on the encoding and the case study

		Feature Selection		Hyperparameters			
					Dense layer parameters		
		Total number of characteristics	Number of selected characteristics	Initial learning rate	Weight initialization	Layer Size	Activation Function
CAF	SouFNN	65	64	0.0254	Glorot normal	214	ReLU
	ExtFNN	103	88	0.0303	Normal	256	ReLU
	MapFNN	6	5	0.0760	Normal	256	Hardtanh
	HumFNN	12	10	0.0397	Normal	152	Hardsigmoid
BSH	SouFNN	13	11	0.0882	Normal	206	ReLU
	ExtFNN	21	19	0.0019	Glorot normal	211	Hardsigmoid
	MapFNN	6	4	0.0041	Normal	202	Sigmoid
	HumFNN	6	6	0.0603	Glorot normal	199	ReLU

4.5.1 Linguistic-baseline: Linguistic Rule-Based Baseline

Spanoudakis et al. [84] propose automatically generating the traces between requirements and models. To do this, they present a linguistic rule-based approach with the following two stages:

- Stage 1) A Parts-of-Speech tagging technique [52] is applied on the requirements that are defined using natural language.
- Stage 2) The traceability links between the requirements and the models are generated through the *Requirement-To-Object-Model* (RTOM) rules.

In [84], there are two different types of traceability rules: RTOM for traceability relations between requirements and model elements, and inter-requirement rules for traceability relations between different parts of a requirement statement. The RTOM rules are specified by investigating grammatical patterns in requirements. Fig. 5 shows an example of a RTOM rule for each case study.

The rule for the CAF case study in Fig. 5 attempts to match a syntactic expression that consists of a noun (<x1/{NN1, NN2}>), the verb *to be* in the present form (<x2/{VBZ, VBR}>), and an adjective (<x3/{JJ}>) with an attribute in the model. The matching succeeds if: (a) the name of the attribute contains the adjective and the name of the class that defines the attribute contains the noun; or (b) the name of the attribute contains the adjective and the name of the class that defines the attribute contains the singular form of the noun. Therefore, in Fig. 6, the sequence of terms <NN1>door</NN1> <VBZ>is</VBZ>

Table 4 RNN setup for the FLiM-ML approach depending on the encoding and the case study

		Feature Selection		Hyperparameters			
						LSTN layer parameters	
		Total number of characteristics	Number of selected characteristics	Initial learning rate	Weight initialization	Layer Size	Activation Function
CAF	SouRNN	65	63	0.0567	Glorot normal	215	Sigmoid
	ExtRNN	103	93	0.0907	Glorot normal	192	Tanh
	MapRNN	6	6	0.0110	Glorot normal	153	Sigmoid
	HumRNN	12	10	0.0633	Glorot normal	182	Sigmoid
BSH	SouRNN	13	12	0.3333	Normal	256	ReLU
	ExtRNN	21	19	0.0413	Normal	165	ReLU
	MapRNN	6	4	0.0792	Normal	241	HardTanH
	HumRNN	6	4	0.0445	Glorot normal	247	ReLU

Fig. 5 Example of a *requirement-to-object-model* rule for each case study

```
RTOM_RULE Rule-CAF:
EXISTS
   SEQUENCE(<x1/{NN1, NN2}>,<x2/{VBZ, VBR}>,<x3/{JJ}>) in Requirement;
   <x4/CLASS>, <x5/ATTRIBUTE> in Model
SUCH THAT
   ATTRIBUTE_OF(<x5>,<x4>) and CONTAINS(NAME(<x5>), <x3>) and (CONTAINS(NAME(<x4>), <x1>) or
   CONTAINS(NAME(<x4>), SINGULAR_FORM<x1>)
ACTION GENERATE
   OVERLAPS(Requirement, <x5>)
RTOM_RULE_END
```

```
RTOM_RULE Rule-BSH:
EXISTS
   SEQUENCE(<x1/{NN1, NN2}>,<x2/{VBN}>,<x3/{PRT}>,<x4/{CD}>,<x5/{JJ}>,<x6/{NN1, NN2, NNS}>)
   in Requirement;
   <x7/CLASS>, <x8/CLASS> in Model
SUCH THAT
   ASSOCIATION_OF(<x7>,<x8>) and CONTAINS(NAME(<x7>),<x1>))
   and CONTAINS(NAME(<x8>), SINGULAR_FORM<x6>)
ACTION GENERATE
   OVERLAPS(Requirement,<x7>), OVERLAPS(Requirement,<x8>)
RTOM_RULE_END
```

<JJ>closed</JJ> in the requirement and the attribute Closed of the class Door satisfy the conditions of the rule. As a consequence, an *Overlap* relation is created between them.

The rule for the BSH case study in Fig. 5 attempts to match a syntactic expression with two classes that are associated in the model. The syntactic expression consists of a noun (<x1/{NN1, NN2}>), a verb (<x2/{VBN}>) followed by "to" (<x3/{TO}>), a cardinal number (<x4/{CD}>), an adjective (<x5/{JJ}>), and a noun that can be singular or plural (<x6/{JJ}>). The matching succeeds if: the name of a class contains the first noun, the name of the other class contains the singular form of the second noun, and both classes are associated in the model. For example, for the requirement "*Inductor connected to two internal inverters an*", this rule will create *Overlap* relations between the requirement and the classes Inductor and Inverter.

Furthermore, Fig. 6 shows how a RTOM rule is applied to obtain a grammatical pattern between a requirement and a model. To do this, the example in this figure is based on the rule for the CAF case study (see Fig. 5). The pattern to find in the requirement is composed of a noun followed by a verb and an adjective. When this pattern is found, the sequence of terms (e.g., <NN1>door</NN1> <VBZ>is</VBZ>

<JJ>closed</JJ>) is linked to the model. Specifically, it is linked to a class whose name must be the noun (i.e., Door) and its attribute must be the adjective (i.e., Closed).

The authors in [84] propose 26 rules for two domains: a software-intensive TV system created by Philips, and a university course management system. Some of these rules are RTOM (i.e., trace relations between requirements and model elements) and some of them are inter-requirement (i.e., traceability relations between different parts of a requirement statement). However, only the total number of rules is specified in their work. Given the two types of rules and the two domains, we estimated that six RTOM rules and six inter-requirement rules were defined at least for each domain. Since our approach focuses on feature location in models, only RTOM rules are tackled in this work. Therefore, based on the guides and the examples of rules that are provided by [84], a domain expert who was not involved in the research generated the initial set of RTOM rules. The initial sets (one for each domain) had a least six rules as the reference work.

Nevertheless, our work has two main differences with respect to the work in [84]. First, the query is a feature description instead of a requirement. Second, the domains are different from the ones evaluated in [84]. Due to these differences, we considered that the number of rules could not be enough to achieve good results in our work. To mitigate this problem, a second expert who was not involved in the research extended the sets of rules. Thus, we not only ensured the suitability of the rule sets for our evaluation, but also mitigate the dependence on a single domain expert. In the end, the extended set for the CAF domain contains nine RTOM rules and the extended set for the BSH domain contains eight rules.

4.5.2 IR-baseline: Information Retrieval Baseline

Information Retrieval (IR) [30,57,81] is a sub-field of computer science that deals with the automated storage and retrieval of documents. IR techniques have been successfully used to retrieve traceability links between different kinds of software artifacts in different contexts [56,70,1,61,23]. In [22] and [55], De Lucia et al. use Latent Semantic Indexing (LSI) to recover traceability links between requirements and different kinds of software artifacts, including models in the form of use-case diagrams, among others. We use LSI to recover traceability links between requirements and models as one of the approaches for our evaluation.

Latent Semantic Indexing (LSI) [47] is an automatic mathematical/statistical technique that analyzes relationships between queries and documents (bodies of text). LSI constructs vector representations of both a user query and a corpus of text documents by encoding them as a *term-by-document co-occurrence matrix* and analyzes the relationships between those vectors to get a similarity ranking between the query and the documents (see Fig. 7).

In feature location, the query corresponds to the feature description and each document is a natural language representation of a model element that is extracted using the technique in [66]. The top of Fig. 7 shows an example of a *term-by-document co-occurrence matrix*, with values associated with our real case. Each row in the matrix (keywords) represents each of the words that compose the query and the documents (e.g., pantograph or door). Each column in the matrix represents a document, and the final column represents the query. Each cell in the matrix contains the frequency with which the term (keyword) of its row appears in the document denoted by its column. For instance, in Fig. 7, the term 'pantograph' appears twice in the document of the second model element (ME2) and once in the query.

Vector representations of the documents and the query are then obtained by normalizing and decomposing the *term-by-document co-occurrence matrix* using a matrix factorization technique called Singular Value Decomposition [47]. The bottom of Fig. 7 shows a three-dimensional graph of the Singular Value Decomposition technique. For legibility reasons, only a small set of the columns is represented. To measure the degree of similarity between vectors, the cosine between the query vector and the document vectors is calculated. Cosine values that are closer to 1 denote a higher degree of similarity, and cosine values that are closer to -1 denote a lower degree of similarity. Similarity increases as vectors point in the same general direction (as more terms are shared between documents). With this measurement, the model elements are ordered according to their degree of similarity to the feature description (see Fig. 7, bottom left).

From the ranking, of all the model elements, only those model elements that have a degree of similarity greater than x must be taken into account. A good heuristic that is widely used is $x = 0.7$. This value corresponds to a 45° angle between the corresponding vectors. Even though the selection of the threshold is an issue under study, the heuristic chosen for this work has yielded good results in other similar works [62,80].

Following this principle, the elements with a degree of similarity equal to or greater than $x = 0.7$ are taken to conform a model fragment, which is a candidate for realizing the feature. In the example provided in Fig. 7, ME2 and MEN are model elements that conform part of the model fragment that is obtained by this baseline for the feature because their cosine values are greater than the 0.7 threshold. The model fragment generated in this manner is the final output of the IR-baseline.

4.6 Implementation details

We used the Eclipse Modeling Framework to manipulate the models and to manage the model fragments. RankBoost was implemented using the RankLib library [21].

Fig. 6 Example of a grammatical pattern for RTOM rules

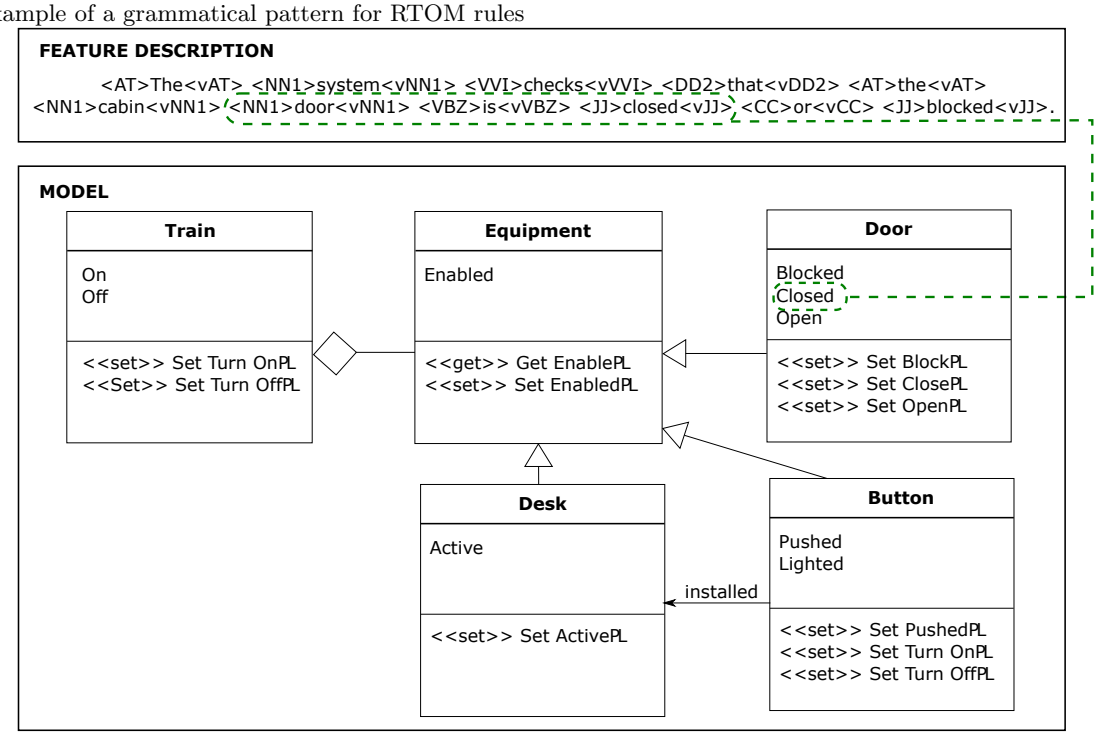
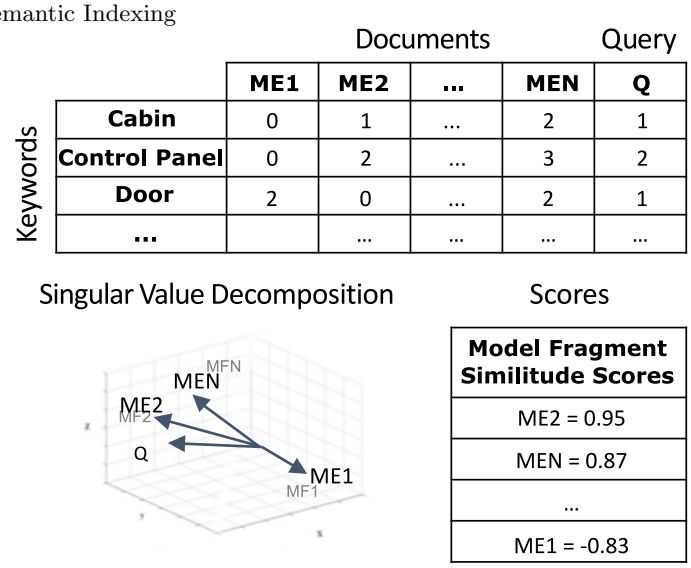


Fig. 7 Example of Latent Semantic Indexing



The neural networks were developed and tuned through the Deep Learning for Java library [86]. Finally, the classifiers were validated through the k-fold validation method [38] with a k value equal to 4. Moreover, the folds were selected in a stratified manner so that each partition contains roughly the same proportions of each score value.

To replicate the results, this work describes the parameters for each one of the machine learning techniques in Tables 2,3, and 4. The implementation for the approach is available at `http://bitbucket.org/svitusj/flame/src/master/FLiM_ML/`. The implementation for the four encodings (source encoding, extended encoding, mapped encoding, and human encoding) is also available at the same location within the implementation of the approach. We have also made the classifiers and a test case available at the same url. Therefore, our public online repository contains the source code of the machine learning approach, the source code of the four encodings, the classifiers for all the trainings, and one of the test cases. Furthermore, the implementation for the Information Retrieval approach and the Linguistic rule-based approach is also available in the same repository under the names of TLR-LSI and TLR-Linguistic, respectively.

4.7 Results

In Table 5, we outline the results of the FLiM-ML approach for each possible combination of encoding and machine learning technique. The different combinations are named using acronyms. The first three letters of the acronyms indicates the encoding used: source encoding (Sou), extended encoding (Ext), mapped encoding (Map), or human encoding (Hum). The rest of the acronym indicate the machine learning technique used: RankBoost (Rank), Feedforward Neural Network (FNN), or Recurrent Neural Network (RNN). Each row shows the Precision, Recall, F-measure, and MCC values obtained applying the approach for all the possible combinations.

RQ1 involves the results of the FLiM-ML approach when it uses the source encoding, the extended encoding, and the mapped encoding with RankBoost as the machine learning technique (i.e., SouRank, ExtRank, and MapRank). **In response to RQ1**, the mapped encoding outperforms the results of the other encodings, even the source encoding, when the FLiM-ML approach applies RankBoost. Table 5 shows that the mapped encoding achieves the best results for all the performance indicators in the two case studies. In CAF, FLiM-ML provides a mean precision value of 90.11%, a recall value of 86.20%, a F-measure value of 87.22%, and a MCC value of 0.87. In BSH, FLiM-ML provides a mean precision value of 71.60%, a recall value of 71.30%, a F-measure value of 71.42%, and a MCC value of 0.69.

RQ2 involves the results of the FLiM-ML approach when it uses the source encoding, the extended encoding, and the mapped encoding with RankBoost, FNN, and RNN as the machine learning techniques (i.e., SouRank, ExtRank, MapRank, SouFNN, ExtFNN, MapFNN, SouRNN, ExtFNN, and MapRNN). **In response to RQ2**, the FLiM-ML approach obtains the best results using the mapped encoding and RankBoost. MapRank outperforms the results of the other machine learning techniques regardless of the encoding used. For FNN, the best results are achieved by ExtFNN in the CAF case study and by SouFNN in the BSH case study. For RNN, the best results are achieved by MapRNN in the CAF case study and by SouRNN in the BSH case study. However, Table 5 shows that neither FNN nor RNN outperform the results of MapRank. MapRank achieves the best results for all the performance indicators in the two case studies.

RQ3 involves the results of the FLiM-ML approach when it uses the human encodings and with RankBoost, FNN, and RNN as the machine learning techniques (i.e., HumRank, HumFNN, and HumRNN). These results are compared to the combination of encoding and machine learning, which obtained the best results in RQ1 and RQ2 (i.e., MapRank). **In response to RQ3**, the FLiM-ML approach obtains the best results using the mapped encoding and RankBoost. Therefore, a domain-independent encoding such as MapRank can outperform the results of a domain-dependent encoding such as the human encodings. Taking into account the results, the human encoding for the CAF case study achieves the best precision values when the machine learning technique is RNN (HumRNN) and the best recall, F-measure, and MCC values when the machine learning technique is RankBoost (HumRank). In contrast, the human encoding for the BSH case study achieves the best values for all the performance indicators when the machine learning technique is RankBoost (HumRank). Table 5 shows that the human encodings do not outperform the MapRank, which obtains the best results for all the performance indicators in the two case studies.

Table 6 shows the results, which are aggregated for each of the baselines and for our approach. Our approach uses the combination of encoding and machine learning technique, which obtains the best results for RQ1, RQ2, and RQ3 (i.e., MapRank). Each row shows the Precision, Recall, F-measure, and MCC obtained by each approach for each case study.

In response to RQ4, MapRank achieves the best results for all the performance indicators, providing a mean precision value of 90.11%, a recall value of 86.20%, a combined F-measure value of 87.22%, and a MCC value of 0.87 for the CAF case study. For the BSH case study, the approach provides a mean precision value of 71.60%, a recall value of 71.30%, a combined F-measure value of 71.42%, and a MCC value of 0.69. In contrast, the baselines have the worst results in all the measurements: the Linguistic-baseline obtains up to 39.94% precision, 51,70% recall, 42.74% F-measure, and 0,42 MCC; and the IR-baseline achieves up to 31.40% precision, 56.17% recall, 33.90% F-measure, and 0.32 MCC.

5 Statistical Analysis

To properly compare the different encodings and machine learning techniques, the data resulting from the empirical analysis was analyzed using statistical methods.

5.1 Statistical Significance

A statistical test must be run to assess whether there is enough empirical evidence to claim that there is a difference between two configurations (e.g., A is better than B). To achieve this, two hypotheses are defined: the null hypothesis H_0, and the alternative hypothesis H_1. The null hypothesis H_0 is typically defined to state that there is no difference between the configurations, whereas the alternative hypothesis H_1 states that the configurations differ. In such a case, a statistical test aims to verify whether the null hypothesis H_0 should be rejected.

Table 5 Mean Values and Standard Deviations for Precision, Recall, F-Measure, and Matthews Correlation Coefficient (MCC) for the FLiM-ML approach using the source (Sou), extended (Ext), mapped (Map), and human (Human) encodings with RankBoost (Rank), Feedforward Neural Network (FNN), and Recurrent Neural Network (RNN) for the two case studies (CAF and BSH)

		Precision	Recall	F-measure	MCC
CAF	SouRank	10.95 ± 17.58	55.98 ± 33.39	14.50 ± 18.10	0.03
	ExtRank	10.65 ± 17.15	55.52 ± 33.43	14.22 ± 17.85	0.02
	MapRank	90.11 ± 26.39	86.20 ± 30.17	87.22 ± 28.60	0.87
	HumRank	24.23 ± 34.52	69.45 ± 28.27	28.53 ± 33.68	0.22
	SouFNN	12.17 ± 19.64	26.59 ± 28.09	12.39 ± 16.94	0.04
	ExtFNN	42.04 ± 42.03	41.80 ± 43.38	40.34 ± 42.59	0.35
	MapFNN	30.25 ± 36.23	28.12 ± 35.58	28.14 ± 35.35	0.22
	HumFNN	18.39 ± 28.92	28.85 ± 32.52	17.08 ± 26.85	0.11
	SouRNN	10.71 ± 18.00	21.26 ± 25.03	10.90 ± 15.57	0.02
	ExtRNN	13.12 ± 18.49	18.36 ± 21.97	12.83 ± 16.77	0.05
	MapRNN	13.29 ± 21.97	36.51 ± 31.88	14.62 ± 20.60	0.05
	HumRNN	28.89 ± 35.77	25.59 ± 35.34	25.49 ± 34.43	0.20
BSH	SouRank	11.72 ± 18.91	10.63 ± 18.71	11.05 ± 18.74	-0.22
	ExtRank	11.07 ± 16.74	9.36 ± 15.83	9.97 ± 16.07	-0.25
	MapRank	71.60 ± 42.89	71.30 ± 43.10	71.42 ± 43.00	0.69
	HumRank	12.41 ± 24.18	10.65 ± 21.58	11.32 ± 22.46	-0.12
	SouFNN	19.24 ± 28.00	18.84 ± 28.84	18.75 ± 27.90	0.02
	ExtFNN	16.34 ± 33.08	13.23 ± 29.00	14.10 ± 29.70	0.05
	MapFNN	8.75 ± 15.04	7.87 ± 15.89	7.97 ± 14.41	-0.14
	HumFNN	8.52 ± 23.41	7.63 ± 20.85	7.83 ± 21.20	0.00
	SouRNN	16.23 ± 34.74	15.06 ± 33.42	15.31 ± 33.42	0.10
	ExtRNN	15.83 ± 29.59	13.41 ± 28.22	14.10 ± 28.05	0.08
	MapRNN	8.63 ± 17.53	6.41 ± 13.38	7.13 ± 14.59	-0.12
	HumRNN	12.30 ± 23.31	9.26 ± 17.11	10.16 ± 18.46	-0.07

Table 6 Mean Values and Standard Deviations for Precision, Recall, F-Measure, and Matthews Correlation Coefficient (MCC) for the FLiM-ML approach using the mapped encoding and RankBoost (MapRank), the IR-baseline, and the Linguistic-baseline for the two case studies (CAF and BSH)

		Precision	Recall	F-measure	MCC
CAF	MapRank	90.11 ± 26.39	86.20 ± 30.17	87.22 ± 28.60	0.87
	Linguistic-baseline	39.15 ± 17.11	51.70 ± 20.91	42.74 ± 17.12	0.42
	IR-baseline	21.22 ± 26.95	56.17 ± 40.44	25.27 ± 25.30	0.24
BSH	MapRank	71.60 ± 42.89	71.30 ± 43.10	71.42 ± 43.00	0.69
	Linguistic-baseline	39.94 ± 21.66	46.53 ± 23.11	41.92 ± 22.24	0.41
	IR-baseline	31.40 ± 34.57	55.65 ± 30.03	33.90 ± 28.21	0.32

Statistical tests provide a probability value, $p - Value$. The $p - Value$ obtains values between 0 and 1. The lower the $p - Value$ of a test, the more likely that the null hypothesis is false. It is accepted by the research

community that a $p - Value$ under 0.05 is statistically significant [3], so the hypothesis H_0 is considered false.

The test carried out depends on the properties of the data. Since our data does not follow a normal distribution in general, our analysis required the use of non-parametric techniques [73]. There are several tests for analyzing this kind of data; however, the Quade test is the most powerful one when working with real data [32]. In addition, according to Conover [18], the Quade test is the one that has shown the best results for a low number of configurations.

In our case, we want to compare the results for the encodings and machine learning techniques regarding RQs. Therefore, we have eight different hypotheses, two for each RQ. The null H_0 hypothesis states that there is no difference between the results of the different encodings and techniques. For RQ1, there is no difference between the results of the source encoding, the extended encoding, and the mapped encoding when the approach applies RankBoost (i.e., no difference between SouRank, ExtRank, and MapRank). For RQ2, there is no difference between the results of RankBoost, FNN, and RNN when the approach uses the best encoding for these techniques (i.e., no difference between MapRank, ExtFNN, MapRNNin the CAF case study and no difference between MapRank, SouFNN, and SouRNNin the BSH case study). For RQ3, there is no difference between the results of the best combination of encoding and machine learning in RQ1 and RQ2 and the results of the human encodings (i.e., no difference between MapRank, HumRank, HumFNN, and HumRNN). For RQ4, there is no difference between the results of the best combination of encoding and machine learning in RQ1, RQ2, RQ3 and the results of the baselines (i.e., no difference between MapRank, Linguistic-baseline, and IR-baseline). In contrast, the alternative hypothesis H_1 states that the results of these cases differ.

Table 7 shows the Quade test statistics and $p - Values$ for precision, recall, F-measure, and MCC. Since the $p - Values$ are smaller than 0.05, we rejected the null hypothesis for RQ1, RQ2, and RQ3. Consequently, we can state that there are differences among the results for RQ1, RQ2, and RQ3.

For RQ4, most $p - Values$ are smaller than 0.05, but the $p - Value$ of the recall for the BSH case study is higher than 0.05. Therefore, we can state that there are differences among the results of MapRank and the baselines in the CAF case study. In the BSH case study, we can state that there are differences among the results of MapRank and the baselines for the precision, F-measure, and MCC, but there are no differences for recall.

Nevertheless, with the Quade test, we cannot answer the following question: *Which of the configurations gives the best performance?* In this case, the performance of each configuration should be individually compared with all the other alternatives. To do this, we performed an additional post hoc analysis. This kind of analysis performs a pair-wise comparison among the results, determining whether there are statistically significant differences among the results for each RQ.

Table 8 shows the $p-Values$ of Holm's post hoc analysis according to the RQs. Almost all the $p - Values$ shown in this table are smaller than 0.05, except for the following comparisons: SouRank vs ExtRank for the two case studies, HumRank vs HumRNN for the two case studies, Linguistic-baseline vs IR-baseline for the two case studies, and MapRank vs Linguistic-baseline for BSH, and MapRank vs IR-baseline for BSH. These comparisons have at least one indicator, which is higher than 0.05. Therefore, the rest of the comparisons have significant differences for all the performance measurements.

5.2 Effect Size

Statistically significant differences can be obtained even if they are so small as to be of no practical value [3]. It is then important to assess whether an encoding-technique pair is statistically better than another and to assess the magnitude of the improvement. *Effect size* measures are needed to analyze this.

For a non-parametric effect size measure, we used Vargha and Delaney's $\hat{A}_{12}$ [87]. $\hat{A}_{12}$ measures the probability that running one configuration yields higher values than running another configuration. If the two configurations are equivalent, then $\hat{A}_{12}$ will be 0.5.

For example, $\hat{A}_{12} = 0.7$ means that we would obtain better results in 70% of the runs with the first pair of configurations that have been compared, and $\hat{A}_{12} = 0.3$ means that we would obtain better results in 70% of the runs with the second pair of configurations that have been compared. Thus, we have an $\hat{A}_{12}$ value for every comparison.

Table 9 shows the values of the effect size statistics. In RQ1, the $\hat{A}_{12}$ values indicate that the source encoding and the extended encoding are equivalent, whereas the mapped encoding shows pronounced superiority when it is compared to the source encoding MapRank obtains better results than the source encoding in at least the following percentages of runs: 93% for precision, 77% for recall, 93% for F-measure, and 94% for MCC.

In RQ2, the $\hat{A}_{12}$ values indicate that MapRank is superior to ExtFNN and MapRNN for the CAF case study. MapRank obtains better results than ExtFNN and MapRNN in at least the following percentages of runs: 79% for precision, 76% for recall, 78% for F-measure, and 79% for MCC. In addition, MapRank is also superior to SouFNN and SouRNNfor the BSH case study. It obtains better results than SouFNN and SouRNN in at least the following percentages of runs: 78% for precision, 77% for recall, 78% for F-measure, and 79% for MCC.

Table 7 Quade test statistics and $p - Values$

			Precision	Recall	F-Measure	MCC
CAF	RQ1	p-Value	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$
		Statistic	2852.3	816.83	2667.8	2503.2
	RQ2	p-Value	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$
		Statistic	999.52	754.27	923.32	1163
	RQ3	p-Value	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$
		Statistic	873.8	1398.1	911.98	814.75
	RQ4	p-Value	1.18×10^{-9}	1.3×10^{-4}	4.31×10^{-11}	4.74×10^{-11}
		Statistic	34.88	11.13	44.57	44.14
BSH	RQ1	p-Value	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$
		Statistic	95.67	101.49	102.83	92.90
	RQ2	p-Value	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$
		Statistic	66.29	63.54	66.87	69.34
	RQ3	p-Value	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$	$< 2.20 \times 10^{-16}$
		Statistic	76.09	84.59	83.92	96.87
	RQ4	p-Value	0.030	0.19	7.3×10^{-3}	0.038
		Statistic	4.55	1.86	7.12	4.14

In RQ3, the $\hat{A}_{12}$ values indicate that MapRank is superior to the human encodings regardless of the machine learning technique applied. For the CAF case study, MapRank obtains better results than HumRank HumFNN and HumRNN in at least the following percentages of runs: 86% for precision, 70% for recall, 85% for F-measure, and 86% for MCC. For the BSH case study, MapRank obtains better results than HumRank HumFNN and HumRNN in at least the following percentages of runs: 79% for precision, 80% for recall, 80% for F-measure, and 85% for MCC.

In RQ4, the $\hat{A}_{12}$ values indicate that MapRank is superior to the baselines. Taking into account the results for the two studies, MapRank obtains better results than Linguistic-baseline in at least the following percentages of runs: 77% for precision, 77% for recall, 77% for F-measure, and 76% for MCC. Furthermore, MapRank obtains better results than IR-baseline in at least the following percentages of runs: 72% for precision, 71% for recall, 77% for F-measure, and 76% for MCC. Therefore, MapRank obtains better results than Linguistic-baseline and IR-baseline in at least 71% of the runs for precision, recall, F-measure, and MCC.

6 Discussion

The results reveal that by using the mapped encoding and RankBoost, our approach outperforms the results obtained with the other encodings and machine learning techniques. In fact, by using the mapped encoding and RankBoost, the approach can obtain better results than by using the encoding that is specifically designed to exploit human experience and domain knowledge.

In this section, we discuss the results of this work in comparison to the results obtained in our previous work [60], what prerequisites are needed by the encodings, and what properties are leveraged by each encoding to improve the results.

6.1 Comparison with previous results

The results obtained in this work are different from the results obtained in our previous work [60] using the same encoding (source encoding) and the same machine learning technique (RankBoost). However, the case study, the problem to solve, and the documentation are different. The case study in our previous work consisted of 29 test cases, where the models had about 2000 elements and the requirements had about 50 words. The case study used in this work consists of 1800 test cases, where the models have about 650 elements and the feature descriptions have about 25 words.

In addition to the differences between the case studies, the problem to solve is also different. The previous work focused on traceability link recovery between requirements and models. In contrast, this work focuses on feature location. Since the problem to solve in this work is different, we also needed different documentation from our industrial partner. Instead of requirements, we needed feature descriptions. Requirements are written before development, are client influenced, and are for contracts. In contrast, features are written when products already exist, are internal, and are for reuse. There-

Table 8 Holm's Post Hoc $p - Values$

			Precision	Recall	F-Measure	MCC
CAF	RQ1	SouRank vs ExtRank	0.27	0.68	0.19	0.65
		SouRank vs MapRank	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
		ExtRank vs MapRank	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
	RQ2	MapRank vs ExtFNN	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
		MapRank vs MapRNN	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
		ExtFNN vs MapRNN	$< 2.0 \times 10^{-16}$	6.6×10^{-3}	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
	RQ3	MapRank vs HumRank	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
		MapRank vs HumFNN	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
		MapRank vs HumRNN	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
		HumRank vs HumFNN	2.3×10^{-7}	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	3.7×10^{-13}
		HumRank vs HumRNN	6.3×10^{-5}	$< 2.0 \times 10^{-16}$	9.8×10^{-7}	0.6
		HumFNN vs HumRNN	$< 2.0 \times 10^{-16}$	1.8×10^{-5}	1.9×10^{-12}	2.6×10^{-14}
	RQ4	MapRank vs Linguistic-baseline	4.5×10^{-8}	5×10^{-7}	4.5×10^{-8}	4.5×10^{-8}
		MapRank vs IR-baseline	7.7×10^{-8}	1.2×10^{-3}	4.5×10^{-8}	7.7×10^{-8}
		Linguistic-baseline vs IR-baseline	1.2×10^{-3}	0.46	1.2×10^{-3}	8.4×10^{-4}
BSH	RQ1	SouRank vs ExtRank	0.89	0.95	0.95	0.23
		SouRank vs MapRank	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
		ExtRank vs MapRank	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
	RQ2	MapRank vs SouFNN	$< 2.0 \times 10^{-16}$	2.9×10^{-16}	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
		MapRank vs SouRNN	4.3×10^{-16}	8.7×10^{-16}	$< 2.0 \times 10^{-16}$	1.8×10^{-13}
		SouFNN vs SouRNN	0.015	0.009	0.015	0.008
	RQ3	MapRank vs HumRank	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
		MapRank vs HumFNN	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
		MapRank vs HumRNN	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$	$< 2.0 \times 10^{-16}$
		HumRank vs HumFNN	3.8×10^{-4}	6.5×10^{-4}	9.4×10^{-4}	1.3×10^{-9}
		HumRank vs HumRNN	0.83	0.85	0.88	6.5×10^{-3}
		HumFNN vs HumRNN	3.3×10^{-3}	3.0×10^{-3}	3.4×10^{-3}	3.0×10^{-4}
	RQ4	MapRank vs Linguistic-baseline	0.029	0.048	0.048	0.061
		MapRank vs IR-baseline	0.038	0.17	0.038	0.061
		Linguistic-baseline vs IR-baseline	0.44	0.8	0.44	0.61

fore, requirements and features are written in a different phase of the development, in a different style, and with a different goal in mind.

For all of these reasons, even though the encoding and the machine learning technique are the same as in our previous work, the results are clearly different. In fact, an interesting research question for future work could be to determine which of the three encodings presented in this work obtains the best results for traceability link recovery.

6.2 Encoding prerequisites and properties

The encodings need some prerequisites to be applied, but not all the encodings need the same prerequisites. The source encoding needs an ontology that contains the main concepts and relations of a domain. In contrast, the extended encoding and the mapped encoding need a more complete ontology that contains not only the main concepts and relations, but also the main properties of the concepts. The human encoding does not require an ontology at all.

All the encodings are applied on feature descriptions and model fragments. The feature descriptions must be provided using natural language and the model fragments must conform to MOF (the OMG metalanguage for defining modeling languages). If feature descriptions and model fragments do not satisfy these prerequisites, the encodings could not be applied. However, if the missing prerequisite is the ontology, the human encoding could be applied as long as it is a railway domain or an induction hob domain with similar product models,

Table 9 $\hat{A}_{12}$ statistics

			Precision	Recall	F-Measure	MCC
CAF	RQ1	SouRank vs ExtRank	0.50	0.50	0.50	0.51
		SouRank vs MapRank	0.07	0.23	0.07	0.06
		ExtRank vs MapRank	0.06	0.23	0.07	0.05
	RQ2	MapRank vs ExtFNN	0.79	0.76	0.78	0.79
		MapRank vs MapRNN	0.93	0.85	0.93	0.94
		ExtFNN vs MapRNN	0.65	0.50	0.63	0.66
	RQ3	MapRank vs HumRank	0.86	0.70	0.85	0.86
		MapRank vs HumFNN	0.91	0.87	0.91	0.91
		MapRank vs HumRNN	0.87	0.86	0.87	0.87
		HumRank vs HumFNN	0.61	0.82	0.66	0.59
		HumRank vs HumRNN	0.53	0.83	0.58	0.52
		HumFNN vs HumRNN	0.45	0.55	0.46	0.44
	RQ4	MapRank vs Linguistic-baseline	1	0.93	1	0.99
		MapRank vs IR-baseline	0.95	0.71	0.99	0.99
		Linguistic-baseline vs IR-baseline	0.77	0.44	0.73	0.71
BSH	RQ1	SouRank vs ExtRank	0.51	0.51	0.51	0.54
		SouRank vs MapRank	0.22	0.22	0.22	0.05
		ExtRank vs MapRank	0.21	0.21	0.21	0.05
	RQ2	MapRank vs SouFNN	0.78	0.77	0.78	0.87
		MapRank vs SouRNN	0.79	0.80	0.80	0.79
		SouFNN vs SouRNN	0.63	0.64	0.64	0.34
	RQ3	MapRank vs HumRank	0.79	0.80	0.80	0.91
		MapRank vs HumFNN	0.83	0.84	0.84	0.85
		MapRank vs HumRNN	0.80	0.81	0.81	0.89
		HumRank vs HumFNN	0.63	0.63	0.63	0.26
		HumRank vs HumRNN	0.50	0.49	0.50	0.39
		HumFNN vs HumRNN	0.37	0.38	0.38	0.63
	RQ4	MapRank vs Linguistic-baseline	0.77	0.77	0.77	0.76
		MapRank vs IR-baseline	0.72	0.71	0.77	0.76
		Linguistic-baseline vs IR-baseline	0.70	0.38	0.66	0.63

or an ontology could be defined from the experience of the company employees.

The success of the mapped encoding lies in the properties of the encodings (length, completeness, domain independence). Even though we have all the necessary artifacts to apply an encoding, the results may not be as good as we expect because the design of each encoding favors some properties over others. However, the mapped encoding benefits from all of them.

Length refers to the number of characteristics used to encode a model fragment and a feature description into a feature vector. In our case studies, the source encoding, the extended encoding, and the mapped encoding contain 65, 103, and 6 characteristics, respectively. Moreover, the human encoding for the railway domain and the human encoding for the induction hob domain contain 12 and 8 characteristics, respectively.

A lower number of characteristics usually leads to better learning performance, lower computational cost, and better classifier interpretability [69,94]. For these reasons, an encoding with a shorter length may obtain better results than an encoding with a longer length. The mapped encoding has the lowest length.

Completeness refers to the degree of having all the necessary information or having nothing missing. The completeness of the source encoding depends on the concepts and relations of the ontology. The extended encoding and the mapped encoding depend on the concepts, properties, and relations of the ontology. The completeness of the human encodings depends on human capacity and experience to propose the characteristics.

The knowledge base is a large source of information to learn how to locate features. However, the way this source should be encoded depends on the encoding. The encoding decides what terms must be counted in the feature description or what elements must be considered in the model fragments. Missing information can lead to poor learning performance. For this reason, an encoding with higher completeness may obtain better results than an encoding with lower completeness. The mapped ending is based on all the concepts, relations, and properties of a domain, so it has an advantage in comparison to the source encoding, which is only based on concepts and relations. Moreover, the completeness of the human encodings depends on human decisions. If domain experts do not consider that a concept (e.g., pantograph) is relevant, it will not be included in the human encoding. In contrast, for the mapped encoding, if the concept is in the ontology, it will always be included. This can lead to less completeness in the human encodings in comparison to the mapped encoding.

Domain independence refers to the extent to which encoding can be generalized to other domains. The source encoding, the extended encoding, and the mapped encodings are domain independent, so they can be applied in other domains provided that there is an ontology. In contrast, the human encoding is specifically designed for a certain domain, so it cannot be applied to other domains.

The domain independence could be negative or positive depending on the perspective. On the one hand, a domain-dependent encoding can exploit human knowledge and domain knowledge providing higher completeness of the encoding. However, this encoding must be done for each domain, so there is an additional cost related to its design for each specific domain. On the other hand, a domain-independent encoding can be generalized to other domains, so other domains may benefit from the same encoding. Nevertheless, the domain knowledge that is included through the encoding may not be as complete as in a domain-dependent encoding.

For example, the human encoding, which is a domain-dependent encoding, takes into account the *Orders* elements. In the models of our case study, an *Order* is a model element that describes an action of another model element. For the model fragment in Fig. 2, an *Order* of a *Door* element may change the door state from open to closed. However, the *Orders* are not included as part of the ontology, so the encodings based on the ontology (the source encoding, the extended encoding, and the mapped encoding) miss this kind of domain knowledge. Nevertheless, these encodings only need an ontology to be applied in other domains.

For these reasons, RQ3 compares a domain-independent encoding (the mapped encoding) with a domain-dependent encoding (the human encoding). By means of this comparison, we want to determine if a domain-independent encoding (that misses part of the domain knowledge) can provide better results than a domain-dependent encoding that exploits not only the domain knowledge but also the human experience. In our case, the human encoding was designed by two domain experts who have wide experience in the domain, but the results should be replicated in other domains before assuring their generalization.

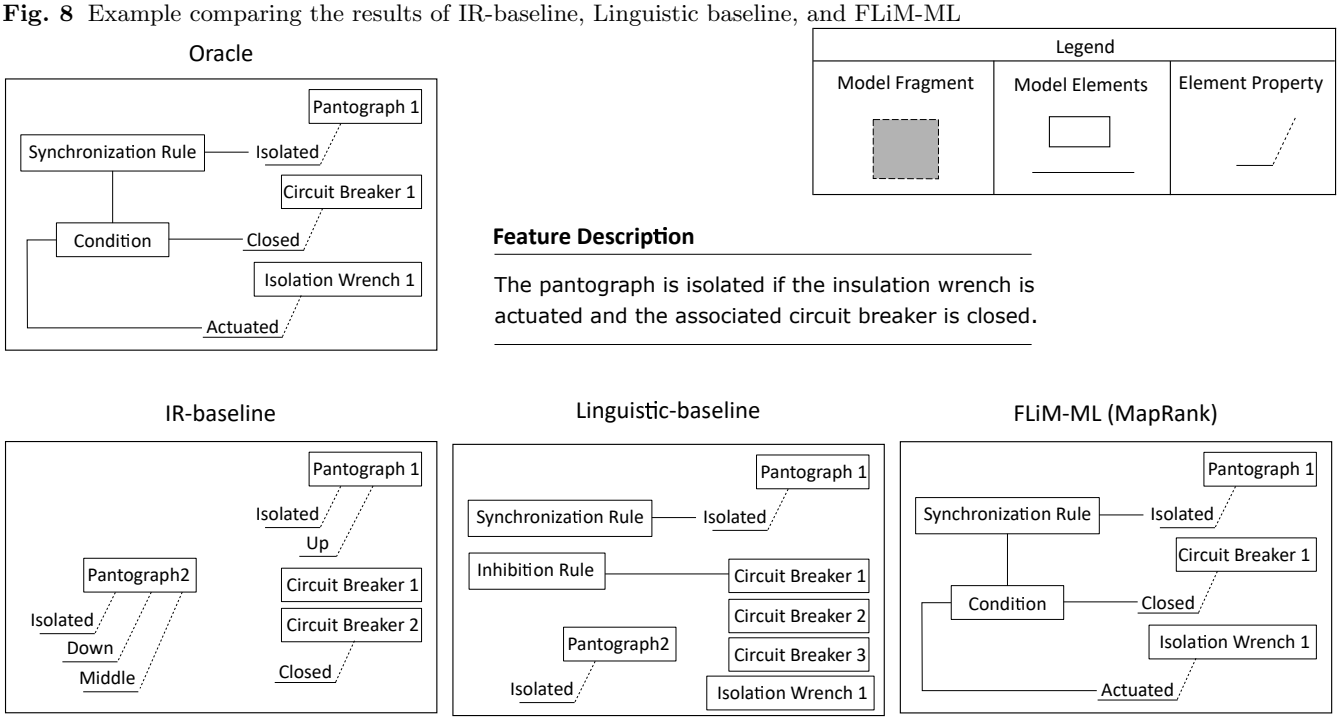
6.3 Nature of the feature descriptions and the model fragments

Both feature descriptions and model fragments use natural language, but they use natural language in a different way. While the feature descriptions are completely based on natural language, the model fragments are based on a meta-model. Therefore, although the model fragments use natural language to define the names of the elements, these elements follow the meta-model structure. This difference in the nature of the feature descriptions and the model fragments causes different ratios between the terms in the feature description and the elements in the model fragments.

For example, in Fig. 2, the concept *Button* appears once in the feature description and there is one element with the name *Button* in the model fragment. In this example, there is a one-to-one ratio for the concept *Button*. However, since natural language is used to describe the feature, the feature may be described in a different way by other domain experts. For example, the feature description could be described as follows: *The state of the doors on one side of the train is controlled by a button. The system turns on the LED of this button if all the doors of the correspondent side are closed or blocked.* In this case, the concept *Button* is repeated twice in the feature description, but it does not mean that there are two buttons in the model fragment. In fact, the number of elements in the model fragment is equal, so there would be a two-to-one ratio for the concept *Button*.

In our work, the only encoding that considers this dependency between the characteristics is the mapped encoding. The source encoding, extended encoding, and human encodings encode the feature descriptions and the model fragments separately. For this reason, we did not consider machine learning techniques such as Naïve Bayes, whose main assumption is the conditional independence of the characteristics (i.e., all characteristics are independent given the value of the class variable). Even if Naïve Bayes also shows good performance when

Fig. 8 Example comparing the results of IR-baseline, Linguistic baseline, and FLiM-ML



there are dependencies between characteristics, the performance can decrease when there is a strong correlation between two or more characteristics, such as in the source encoding, the extended encoding, and the human encodings.

In fact, the good results of the mapped encoding come in part from RankBoost. RankBoost is able to do well on data sets of varying sizes [31]. It can benefit from a small training dataset together with a small number of characteristics in the encoding to reduce the overfitting problem [95,88]. Our training datasets for the CAF and the BSH case studies are composed of 1339 and 758 feature vectors, respectively. This fact and the reduced number of characteristics in the mapped encoding result in the good results of RankBoost. However, the neural networks usually require larger training datasets. For example, for a similar research problem in [35], the training dataset is composed of 45% of the 769,366 artifacts, so it contains about 423,151 feature vectors. Therefore, while RankBoost benefits from the size of the training datasets in our studies, these datasets are not enough for the neural networks.

Therefore, a theoretical evaluation on the evolution of performance metrics by the number of feature vectors in training set would be useful to determine the applicability of a machine learning technique over others. However, this evaluation is not as straightforward as might suppose at first glance. The number of feature vectors is very important for the training. However, the content of the feature vectors also influences the training. Training with large model fragments and trying to locate a small model fragment is not the same as the opposite. According to [7], the content of the feature vectors (i.e. the model fragments) could also influence the training and ultimately affects the results of Rankboost and the neural networks.

6.4 Traditional Approaches

The tacit knowledge and the vocabulary mismatch problems have a special impact on the approaches that are based on textual similarity. The tacit knowledge is caused by the lack of written information in the feature descriptions. Often, when feature descriptions are written, part of the domain knowledge related to the features is not embodied in their descriptions. Since it is assumed that all the domain experts known this information, it is never formalized in writing.

The vocabulary mismatch is caused by the use of different terms to reference the same concept. In industrial environments, sometimes the feature descriptions and the models are created and manipulated by different engineers. Therefore, the same concept can be referred to using a name in the feature description and a different name in the model.

Both Linguistic-baseline and IR-baseline base, to a large extent, on the textual similarity. IR-baseline compares the text in the feature descriptions and the text in the models according to the co-occurrences of terms in them. Linguistic-baseline compares the text in the feature descriptions with the elements in the models according to grammatical patterns. In both cases, the lack of terms that is caused by the tacit knowledge makes it

impossible to locate the elements from the models that are relevant to the feature descriptions.

In contrast, the vocabulary mismatch problem can be minimized using natural language processing to homogenize the terms in feature descriptions and model fragments. However, the in-house terms that are used in a specific domain or company are not known synonyms. For this reason, this problem can have an impact on results, even if the natural language processing methods are applied.

In contrast to Linguistic-baseline and IR-baseline, FLiM-ML is not based on textual similarity. FLiM-ML depends on the learning from the manually located features. Moreover, although the text of the feature descriptions and the model fragments is used by the encodings, the encodings also consider the structure of the model fragment (i.e., relation between elements). For these reasons, FLiM-ML is less sensitive to the tacit knowledge and the vocabulary mismatch problems, providing better results than Linguistic-baseline and the IR-baseline.

Fig. 8 shows an illustrative example to compares the results of the three approaches: IR-baseline (bottom-left), Linguistic-baseline (bottom-middle), and FLiM-ML (bottom-right). The top-left part of Fig. 8 shows the oracle, which is the correct model fragment for the feature description in the figure. Comparing the three results to the oracle, we can see that the IR-baseline and the Linguistic-baseline retrieved several model elements of types: *Pantograph* and *Circuit Breaker*. This leads us to suppose that baselines have difficulties to differentiating between two elements of the same type (e.g., between two pantographs). In the example, the feature description does not provide explicit details that allow differentiating between the pantograph to be located and the rest of the pantographs. For this reason, an approach that is not based only on textual similarity, like FLiM-ML, can obtain better precision results. FLiM-ML learns how to differentiate the model elements of the same type thanks to the training using the manually located features.

In addition, the results in Fig. 8 also show that the IR-baseline did not retrieve the model element of type *Synchronization Rule*, but the Linguistic-baseline retrieved it. The term *Synchronization Rule* does not appear in the feature description, so the IR-baseline cannot use this term for the textual similitude. However, the Linguistic-baseline can identify a syntactic pattern in the feature description (e.g., <NN>Pantograph</NN> <VBZ>is</VBZ> <JJ>isolated</JJ> <IN>if</IN>) that matches an element in the model (e.g., *Synchronization Rule*). Therefore, although the name of the model element is not explicit in the feature description, the Linguistic baseline can locate it.

Nevertheless, neither the IR-baseline, nor the Linguistic baseline perfectly located the model fragment in the example. The difference between two model elements of the same type is not explicitly in the feature description. In addition, the meaning of "*associated circuit breaker*" is also not included in the feature description. This kind of details is the tacit knowledge which is problematic for the approaches only based on textual similitude.

According to the previous results and discussion, a machine learning-based approach can be better than the traditional approaches as the Linguistic-baseline or the IR-baseline. Nevertheless, a machine learning-based approach, as FLiM-ML, is not the best solution for all the cases. Providing the necessary documentation to successfully apply a machine learning-based is sometimes difficult or not possible. In these cases, a traditional approach, that demands less input documentation to be applicable and does not require training, can be the best solution. For this reason, although the machine learning-based approach can achieve better results than traditional approaches, we should consider the trade-off between performance and ease of use to answer RQ4.

7 Threats to Validity

In this section, we use the classification of threats to validity of [91] to acknowledge the limitations of our approach. Table 10 shows the threats that are applicable to our evaluation. In Table 10, each type of threat is organized into two groups: avoided (the risk of the threat has been removed) and reduced (the risk of the threat has been minimized). The last column of Table 10 shows how the threats have been dealt with.

Table 10: Threats to Validity

Type of threat	Status	Threat	Due to	How we have dealt with it
Conclusion Validity	Avoided	Fishing	Researchers may influence the result by looking for a specific outcome.	We use all the test cases for all the combinations of encoding and machine learning technique run; none of the test cases were removed for any reason whatsoever.
		Reliability of the treatment implementation	The implementation is not similar for different people applying the treatment or for different occasions.	We applied the same approach regardless of the encoding or the machine learning technique used.
	Reduced	Reliability of measures	When you measure a phenomenon twice, the outcome should be the same.	We used four measures: precision, recall, F-measure, and MCC, which are widely accepted in the software engineering research community. As suggested by [4], several repetitions were performed to obtain reliable results. Each test case was run 30 times.
		Lower statistical power	Sample size is not enough.	We used 1800 different test cases and statistically analyzing the results.
		Random heterogeneity of subjects	Subjects are randomly selected and they are too heterogeneous.	We selected different model fragments and grouped them taking into account their heterogeneity with regard to the measurements: density, dispersion, and multiplicity.
Internal Validity	Avoided	Resentful demoralization	Subjects receiving less desirable treatments may give up and not perform as well as they generally do.	We tuned the parameters to maximize the performance of all the techniques and to perform a fair evaluation.
		Selection	The outcomes can be affected by how the subjects are selected.	We used balanced knowledge bases, which contain samples with high and low scores.
	Reduced	Instrumentation	Effect caused by the artifacts used for experiment execution.	We validated the ontology with different domain experts to check that the ontology was well designed so that it did not negatively affect the experiment.
External Validity	Avoided	Interaction of selection and treatment	This is the effect of not having the experimental setting or material representative of industrial practice.	We used the most recent version of the Eclipse Modeling Framework to perform the implementation.
	Reduced			We dealt with this threat in two ways, by (1) using formats that are frequently leveraged to specify all kinds of different software (e.g., MOF) and (2) designing and developing the approach independently of the domain. Nevertheless, the experiment and its results should be replicated in more domains before assuring their generalization.

Construct validity	Avoided	Interaction of different treatments	There is no way to conclude whether the effect is due to any of the treatments or to a combination of treatments.	We separated the research into encodings and machine learning techniques using RQs. Specifically, for RQ1, we used the same machine learning technique, so only the encodings are involved in answering RQ1. For RQ2, we used the best encoding for each technique, so only the machine learning techniques were involved in answering RQ2. For RQ3, we used the same encoding, so only the machine learning techniques were involved in answering RQ3. In no case were the encodings evaluated at the same time as the machine learning techniques, so there was no interaction between them.

8 Related Work

There are a number of approaches that can improve developers' effectiveness in locating features, which are largely based upon source code [25,17,19,34,80,77]. For feature location in source code, textual analysis is usually performed using three main techniques: pattern matching, information retrieval, and natural language processing [25,17]. Pattern matching algorithms are usually used to locate a pattern (also called strings) in a source code through utilities such as the grep[5] tool. These utilities are mostly based on simplistic string pattern detection (e.g., LIKE in SQL). Information retrieval techniques, such as latent semantic indexing [24], latent dirichlet allocation [11], and vector space model [81], are statistical methods that are used to locate a feature taking into account the textual similarity between source code documents and a query provided by a user. Natural language processing approaches can also exploit a query, but they analyze the parts of speech of the words used in source code [25].

Most of the approaches for feature location in source code are based on machine learning techniques. The work in [19] explores the use of a particular deep learning model, document vectors, for feature location. In [34], a novel deep neural network that embeds code snippets and natural language descriptions into a high-dimensional vector space is proposed. The work in [80] investigates the results of using an agglomerative hierarchical clustering algorithm to identify code-topics. In [77], the high dimensionality of the feature space is reduced by applying latent dirichlet allocation, and K-means clustering is used to cluster the related components of the software system.

However, all these works focus on feature location in source code. Since our work focuses on feature location in models, the models are the main software artifacts. Models raise the abstraction level using concepts that are much less bound to the underlying implementation and technology and are much closer to the problem domain [12]. There are only a few approaches that can locate features based upon other artifacts such as models [48, 29,92,93,89,42,64].

Some of these approaches are also based on information retrieval techniques and natural language processing techniques. Lapeña et al. [48] analyze the impact of the natural language processing techniques when they are used to process feature descriptions and software artifacts for feature location in models. In [29], the approach combines genetic algorithms and information retrieval techniques to locate the features in models.

Other works focus on comparisons among the models for feature location in models. In [92], the feature is located by automatically comparing the models to find the common and the variable elements. The approach proposed by Zhang et al. [92] is refined in [93] in order to reduce the manual effort required in the formalization of the feature realizations when new product models are included in a product line. Wille et al. [89] propose an approach to identify the variability between models, which is based on an exchangeable metric for different attributes of the models. Holthusen et al. [42] present an improved approach for family mining that compares blocks to determine the similarity between models. Martinez et al. [64] compare a set of model variants and identify commonality and variability in the form of what is referred to as features.

All these approaches are based on the location of features through different methods: information retrieval, natural language processing, and comparisons of the models. In contrast, our approach is based on machine learning techniques, which allows resources such as the manually located features that companies have gathered to be exploited.

Moreover, there are some approaches that are based on evolutionary algorithms [29,28]. Since the search spaces (product models) are so large, it is impossible

[5] https://www.gnu.org/software/grep/manual/grep.html

to thoroughly explore the space for possibilities. Therefore, these approaches benefit from the evolutionary algorithms to efficiently explore the possibilities (model fragments). To guide the evolutionary algorithms, these approaches use information retrieval techniques such as latent semantic indexing. Thus, the information retrieval techniques determine which model fragment is the best realization of a feature.

In our work, we do not tackle the exploration of the search spaces (product models); we only focus on determining which model fragment is the best realization of a feature. To do this, instead of using information retrieval techniques, we use machine learning techniques. This work focuses specifically on the identification of a software model encoding for feature location approaches.

In order to find the recent works related to software model encodings, we conducted a limited informal search of the literature. The query that was used to look for related works was: *What are the characteristics usually used to encode software models in model driven engineering?* The search string used was the following: "machine learning" AND ("software model" OR "model driven development" OR "model driven engineering") AND ("encoding" OR "feature vector" OR "model embedding"). The inclusion criterion was (IC1) papers that describe the characteristics used. The exclusion criterion was (EC1) papers that do not understand the term *model* as an abstraction, e.g., the papers where the term *model* means a reference used as an example to copy, follow, or imitate. The search was run in December 2020 on Scopus and ArXiv taking into account the title, keywords, and abstract of the articles. We also completed the search taking into account the works manually found, which satisfy the inclusion and exclusion criteria. The works selected from the set of papers retrieved by the search were classified based on the type of characteristics used in the encoding:

- Characteristics based on the **shape of the model elements** (e.g., the number of connecting lines, the rectangle size, or the alignment of a shape: vertical or horizontal).
- Characteristics based on the **type of the model elements** (e.g., the number of associations, aggregations, compositions, and generalizations in a class diagram).

8.1 Shape of the model elements

Our search string and search criteria found only one paper by Ho-Quang et al. [41]. They propose 23 image-characteristics and investigate the use of these characteristics for the purpose of classifying Unified Modeling Language (UML) class diagram images. They consider that an automated system with the ability to classify UML class diagram images would be very beneficial for building a corpus of UML models. To do this, their paper specifically aims at providing suitable characteristics and classification algorithms to decide which images should be considered as UML class diagrams and which images should be left out.

The first difference between this work and ours is the type of artifact that is encoded. While they encode images of UML class diagrams, we encode model fragments that conform to MOF and feature descriptions that are described using natural language. The second difference between the two works is the goal. They provide a set of suitable characteristics to identify what images correspond to a UML class diagram; we provide a suitable set of characteristics to locate features in models. Moreover, the two encodings proposed in our work contain characteristics that are based on the type of the model elements instead of being based on the shape of the model elements as in [41].

8.2 Type of model element

Narawita and Vidanage [68] propose a system to obtain elements of use cases and class diagrams. The purpose of their research focuses on the automation of UML diagrams from the analyzed requirement text using natural language processing. To do this, they encode the use cases and the class diagrams using characteristics such as associations between use cases and actors (for the uses cases) or associations between classes, aggregations, compositions, and generalizations (for the class diagrams).

Stikkolorum et al. [85] describe an exploratory study on the application of machine learning for the grading of UML class diagrams. They encoded the diagrams selecting a set of characteristics that are based on the class diagrams' actual values (e.g., class name, attribute name, or multiplicity value) and the UML element type themselves (e.g., class, operation, or association).

Our work differs from [68] and [85] because the encoded artifacts are model fragments and feature descriptions instead of use cases and class diagrams. In addition, the characteristics of encoding used in these works focus on use cases and class diagrams (e.g., associations between use cases and actors or associations between classes), so these characteristics cannot be used to encode other kinds of software models. In contrast, the two encodings proposed in our work depend on an ontology, but they do not depend on the kind of model.

Marcén et al. [60] propose an evolutionary ontological encoding approach to enable machine learning techniques to be used to perform software engineering tasks in models. In that work, the encoding is based on an ontology and is used to encode model fragments and feature descriptions. The encoding proposed in [60] is the source encoding for this work. In fact, [60] is our previous work, which is the starting point of the research and the enhancement of software model encoding.

All the works found in the systematic search of the literature are oriented towards encoding the models as feature vectors. None of them takes into account the possibility of model embedding. The proposed encodings in this work can be considered as input to model embedding in the task of feature location. In addition, word embedding is used to represent words in a continuous and multidimensional vector space, so that it is easy to calculate the semantic similarity between words by calculating the vector distance [53]. Similarly, the text in the feature descriptions and model fragments can be embedded using word embedding techniques. These techniques can even be adapted to tackle the particularities of the models (e.g., structural or behavioral details). These are lines of research that can be explored as future work.

9 Conclusion and future work

In our previous works, we have proposed an approach for feature location in models based on machine learning, providing evidence that machine learning techniques can obtain better results than other retrieval techniques for feature location in models. In feature location in models, the encoding is essential to be able to obtain good results using machine learning techniques. In this paper, we have proposed two new software model encodings and compared them to the source encoding.

The evaluation was based on two real-world case studies, where the best results were achieved by the approach using the mapped encoding and RankBoost. In fact, the approach using the mapped encoding and RankBoost achieves better results than the two traditional approaches in at least 72% of the cases for precision, in at least 71% of the cases for recall, in at least 77% of the cases for F-measure, and in at least 76% of the cases for MCC. Since the mapped encoding is domain independent, it can be used in other software engineering tasks that also perform searches in model fragments, e.g., traceability link recovery or bug location. As our systematic literature search shows, there are scarcely any works about software model encodings and they depend on the type of software model (e.g., UML class diagrams). Our encoding can open the door for more researchers to explore whether machine learning can improve other engineering tasks with software models.

In fact, the promising results of this work lead to interesting research questions for the future, such as the following: *Can we achieve similar results in other domains?*; *Can we locate features in a domain using a different domain for training?*; *Can the neural networks outperform the RankBoost results with a larger knowledge base?*; *Can we obtain better results if we adapt more characteristics from benchmark datasets for research on Learning to Rank?*; *Can we embed the models instead of encoding them?*; *Can a (domain) model created out of the feature descriptions be a successful solution for feature location in models?*. Therefore, in order to answer some of these research questions and to test the external validity of the results of this work, the next steps are clear: 1) other knowledge bases with different sizes must be tested; 2) the encodings must be enhanced through other characteristics or perspectives; 3) other case studies must be tested in other domains.

Acknowledgements This work has been developed with the financial support of the Spanish State Research Agency and the Generalitat Valenciana under the projects DataME TIN2016-80811-P, ALPS RTI2018-096411-B-I00, ACIF/2018/171, and PROMETEO/2018/176 and co-financed with ERDF.

References

1. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering Traceability Links between Code and Documentation. IEEE Transactions on Software Engineering **28**(10), 970–983 (2002)
2. Arcega, L., Font, J., Haugen, Ø., Cetina, C.: An Approach for Bug Localization in Models using two Levels: Model and Metamodel. Software & Systems Modeling pp. 1–26 (2019)
3. Arcuri, A., Briand, L.: A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. Software Testing, Verification and Reliability **24**(3), 219–250 (2014)
4. Arcuri, A., Fraser, G.: Parameter Tuning or Default Values? An Empirical Investigation in Search-Based Software Engineering. Empirical Software Engineering **18**(3), 594–623 (2013)
5. B Le, T.D., Lo, D., Le Goues, C., Grunske, L.: A learning-to-rank based fault localization approach using likely invariants. In: Proceedings of the 25th International Symposium on Software Testing and Analysis, pp. 177–188. ACM, New York, USA (2016)
6. Ballarín, M., Marcén, A.C., Pelechano, V., Cetina, C.: Measures to report the Location Problem of Model Fragment Location. In: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 189–199. ACM, New York, USA (2018)
7. Ballarín, M., Marcén, A.C., Pelechano, V., Cetina, C.: On the influence of model fragment properties on a machine learning-based approach for feature location. Information and Software Technology **129**, 106,430 (2021)
8. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research **13**(Feb), 281–305 (2012)
9. Beyranvand, P., Kucuktezcan, C.F., Cataltepe, Z., Genc, V.M.I.: A Novel Feature Selection Method for the Dynamic Security Assessment of Power Systems Based on Multi-Layer Perceptrons. International Journal of Intelligent Systems and Applications in Engineering **6**(1), 53–58 (2018)
10. Bianchini, M., Maggini, M., Jain, L.C. (eds.): Handbook on Neural Information Processing, *Intelligent Systems Reference Library*, vol. 49, 1 edn. Springer, Verlag Berlin Heidelberg (2013)

11. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. Journal of machine Learning research **3**(Jan), 993–1022 (2003)
12. Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice. Synthesis lectures on software engineering **3**(1), 1–207 (2017)
13. Cai, J., Luo, J., Wang, S., Yang, S.: Feature selection in machine learning: A new perspective. Neurocomputing **300**, 70–79 (2018)
14. Canuto, S.D., Belém, F.M., Almeida, J.M., Gonçalves, M.A.: A Comparative Study of Learning-to-Rank Techniques for Tag Recommendation. Journal of Information and Data Management **4**(3), 453 (2013)
15. Cao, Z., Tian, Y., Le, T.D.B., Lo, D.: Rule-Based Specification Mining Leveraging Learning to Rank. Automated Software Engineering **25**(3), 501–530 (2018)
16. Chandrashekar, G., Sahin, F.: A Survey on Feature Selection Methods. Computers & Electrical Engineering **40**(1), 16–28 (2014)
17. Chochlov, M., English, M., Buckley, J.: A historical, textual analysis approach to feature location. Information and Software Technology **88**, 110–126 (2017)
18. Conover, W.: Practical Nonparametric Statistics, 3rd edn Wiley. New York pp. 250–257 (1999)
19. Corley, C.S., Damevski, K., Kraft, N.A.: Exploring the Use of Deep Learning for Feature Location. In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 556–560. IEEE Computer Society, Washington, USA (2015)
20. Cruz, D., Figueiredo, E., Martinez, J.: A Literature Review and Comparison of Three Feature Location Techniques using ArgoUML-SPL. In: Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems, p. 16. ACM (2019)
21. Dang, V.: The Lemur Project - Wiki - RankLib. `http://sourceforge.net/p/lemur/wiki/RankLib/` (2013). [Online; accessed April-2017]
22. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Enhancing an Artefact Management System with Traceability Recovery Features. In: Proceedings of the 20th IEEE International Conference on Software Maintenance, pp. 306–315. IEEE (2004)
23. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Can Information Retrieval Techniques effectively support Traceability Link Recovery? In: 14th IEEE International Conference on Program Comprehension, pp. 307–316. IEEE (2006)
24. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. Journal of the American society for information science **41**(6), 391–407 (1990)
25. Dit, B., Revelle, M., Gethers, M., Poshyvanyk, D.: Feature location in source code: a taxonomy and survey. Journal of Software: Evolution and Process **25**(1), 53–95 (2013)
26. DL4J: Deeplearning4j Suite Overview. `https://deeplearning4j.konduit.ai/`. [Online; accessed 29-July-2021]
27. Engelbrecht, A.P.: Computational Intelligence: An Introduction, 2nd edn. Wiley Publishing, Chichester, England (2007)
28. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: Feature Location in Model-Based Software Product Lines Through a Genetic Algorithm. In: Proceedings of the 15th International Conference on Software Reuse: Bridging with Social-Awareness, pp. 39–54. Springer, Verlag Berlin Heidelberg (2016)
29. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: Feature location in models through a genetic algorithm driven by information retrieval techniques. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp. 272–282 (2016)
30. Frakes, W.B., Baeza-Yates, R.: Information Retrieval: Data Structures and Algorithms (1992)
31. Freund, Y., Iyer, R., Schapire, R.E., Singer, Y.: An Efficient Boosting Algorithm for Combining Preferences. Journal of machine learning research **4**(Nov), 933–969 (2003)
32. García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced Nonparametric Tests for Multiple Comparisons in the Design of Experiments in Computational Intelligence and Data Mining: Experimental Analysis of Power. Information Sciences **180**(10), 2044–2064 (2010)
33. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256. JMLR.org (2010)
34. Gu, X., Zhang, H., Kim, S.: Deep Code Search. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 933–944. ACM, New York, USA (2018)
35. Guo, J., Cheng, J., Cleland-Huang, J.: Semantically Enhanced Software Traceability using Deep Learning Techniques. In: Software Engineering (ICSE), 2017 IEEE/ACM 39th International Conference on, pp. 3–14. IEEE, IEEE Press Piscataway, New Jersey, USA (2017)
36. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. Journal of machine learning research **3**(Mar), 1157–1182 (2003)
37. Haiduc, S., Bavota, G., Oliveto, R., De Lucia, A., Marcus, A.: Automatic query performance assessment during the retrieval of software artifacts. In: 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pp. 90–99. ACM, New York, USA (2012)
38. Hastie, T., Tibshirani, R., Friedman, J.: The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media (2009)
39. Haugen, Ø., Møller-Pedersen, B., Oldevik, J., Olsen, G.K., Svendsen, A.: Adding standardized variability to domain specific languages. In: 2008 12th International Software Product Line Conference, pp. 139–148. IEEE (2008)
40. Haykin, S.: Neural Networks: a Comprehensive Foundation. Prentice Hall PTR, New Jersey, USA (1994)
41. Ho-Quang, T., Chaudron, M.R., Samúelsson, I., Hjaltason, J., Karasneh, B., Osman, H.: Automatic classification of UML class diagrams from images. In: 2014 21st Asia-Pacific Software Engineering Conference, vol. 1, pp. 399–406. IEEE Computer Society, Washington, USA (2014)
42. Holthusen, S., Wille, D., Legat, C., Beddig, S., Schaefer, I., Vogel-Heuser, B.: Family Model Mining for Function

Block Diagrams in Automation Software. In: 18th International Software Product Lines Conference, pp. 36–43. ACM, New York, USA (2014)
43. Hornik, K., Stinchcombe, M., White, H.: Multilayer Feedforward Networks are Universal Approximators. Neural networks **2**(5), 359–366 (1989)
44. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. rep., DTIC Document (1990)
45. Kıraç, M.F., Aktemur, B., Sözer, H.: VISOR: A Fast Image Processing Pipeline with Scaling and Translation Invariance for Test Oracle Automation of Visual Output Systems. Journal of Systems and Software **136**, 266–277 (2018)
46. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing Neural Networks. In: Advances in neural information processing systems, pp. 971–980. Curran Associates Inc., USA (2017)
47. Landauer, T.K., Foltz, P.W., Laham, D.: An Introduction to Latent Semantic Analysis. Discourse Processes **25**(2-3), 259–284 (1998)
48. Lapeña, R., Font, J., Pastor, Ó., Cetina, C.: Analyzing the Impact of Natural Language Processing over Feature Location in Models. ACM SIGPLAN Notices **52**(12), 63–76 (2017)
49. Lapeña, R., Pérez, F., Cetina, C., Pastor, Ó.: Improving Traceability Links Recovery in Process Models Through an Ontological Expansion of Requirements. In: International Conference on Advanced Information Systems Engineering, pp. 261–275. Springer, Verlag Berlin Heidelberg (2019)
50. Lavesson, N., Davidsson, P.: Quantifying the impact of learning algorithm parameter tuning. In: Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA, vol. 6, pp. 395–400. AAAI Press, California, USA (2006)
51. Lee, K., Kang, K.C., Lee, J.: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In: International Conference on Software Reuse, pp. 62–77. Springer, Verlag Berlin Heidelberg (2002)
52. Leech, G., Garside, R., Bryant, M.: CLAWS4: the Tagging of the British National Corpus. In: Proceedings of the 15th Conference on Computational Linguistics - Volume 1, pp. 622–628. Association for Computational Linguistics (1994)
53. Li, C., Ji, L., Yan, J.: Acronym disambiguation using word embedding. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, pp. 4178–4179 (2015)
54. Liu, D., Marcus, A., Poshyvanyk, D., Rajlich, V.: Feature location via information retrieval based filtering of a single scenario execution trace. In: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, pp. 234–243. ACM, New York, NY, USA (2007)
55. Lucia, A.D., Fasano, F., Oliveto, R., Tortora, G.: Recovering Traceability Links in Software Artifact Management Systems using Information Retrieval Methods. ACM Transactions on Software Engineering and Methodology (TOSEM) **16**(4), 13 (2007)
56. Lucia, D., et al.: Information Retrieval Models for Recovering Traceability Links between Code and Documentation. In: Proceedings of the International Conference on Software Maintenance, pp. 40–49. IEEE (2000)
57. Manning, C.D., Raghavan, P., Schütze, H., et al.: Introduction to Information Retrieval, vol. 1. Cambridge University Press (2008)
58. Marcén, A.C., Font, J., Pastor, O., Cetina, C.: Towards Feature Location in Models through a Learning to Rank Approach. In: Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume B, Sevilla, Spain, September 25-29, 2017, pp. 57–64. AC, New York, U (2017)
59. Marcén, A.C., Lapeña, R., Pastor, Ó., Cetina, C.: Traceability link recovery between requirements and models using an evolutionary algorithm guided by a learning to rank algorithm: Train control and management case. Journal of Systems and Software p. 110519 (2020)
60. Marcén, A.C., Pérez, F., Cetina, C.: Ontological Evolutionary Encoding to Bridge Machine Learning and Conceptual Models: Approach and Industrial Evaluation. In: International Conference on Conceptual Modeling, pp. 491–505. Springer, New York, USA (2017)
61. Marcus, A., Maletic, J.I.: Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing. In: Proceedings of the 25th International Conference on Software Engineering, pp. 125–135. IEEE (2003)
62. Marcus, A., Sergeyev, A., Rajlich, V., Maletic, J.: An Information Retrieval Approach to Concept Location in Source Code. In: Proceedings of the 11th Working Conference on Reverse Engineering, pp. 214–223 (2004). DOI 10.1109/WCRE.2004.10
63. Martinez, J., Ziadi, T., Bissyandé, T.F., Klein, J., Traon, Y.L.: Bottom-up Adoption of Software Product Lines: a Generic and Extensible Approach. In: Proceedings of the 19th International Conference on Software Product Lines, pp. 101–110. ACM, New York, USA (2015)
64. Martinez, J., Ziadi, T., Klein, J., Le Traon, Y.: Identifying and visualising commonality and variability in model variants. In: European Conference on Modelling Foundations and Applications, pp. 117–131. Springer (2014)
65. Matthews, B.W.: Comparison of the predicted and observed secondary structure of t4 phage lysozyme. Biochimica et Biophysica Acta (BBA)-Protein Structure **405**(2), 442–451 (1975)
66. Meziane, F., Athanasakis, N., Ananiadou, S.: Generating Natural Language Specifications from UML Class Diagrams. Requirements Engineering **13**(1), 1–18 (2008)
67. Moreira, C., Calado, P., Martins, B.: Learning to Rank Experts in Academic Digital Libraries. In: 15th Portuguese Conference on Artificial Intelligence, EPIA, Lisbon, Portugal (2011)
68. Narawita, C.R., Vidanage, K.: UML generator-an automated system for model driven development. In: 2016 Sixteenth International Conference on Advances in ICT for Emerging Regions (ICTer), pp. 250–256. IEEE Computer Society, Washington, USA (2016)
69. Nie, F., Zhu, W., Li, X.: Unsupervised Feature Selection with Structured Graph Optimization. In: Thirtieth AAAI conference on artificial intelligence, pp. 1302–1308. AAAI Press, California, USA (2016)

70. Oliveto, R., Gethers, M., Poshyvanyk, D., De Lucia, A.: On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery. In: 18th International Conference on Program Comprehension, pp. 68–71. IEEE (2010)
71. Pérez, F., Lapeña, R., Font, J., Cetina, C.: Fragment Retrieval on Models for Model Maintenance: Applying a Multi-objective Perspective to an Industrial Case Study. Information and Software Technology **103**, 188–201 (2018)
72. Poshyvanyk, D., Gueheneuc, Y.G., Marcus, A., Antoniol, G., Rajlich, V.: Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. IEEE Transactions on Software Engineering **33**(6), 420–432 (2007)
73. Potvin, C., Roff, D.A.: Distribution-free and robust statistical methods: viable alternatives to parametric statistics. Ecology **74**(6), 1617–1628 (1993)
74. Qin, T., Liu, T.Y.: Introducing LETOR 4.0 Datasets. Computing Research Repository (CoRR) **abs/1306.2597** (2013)
75. Qin, T., Liu, T.Y., Xu, J., Li, H.: LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval. Information Retrieval **13**(4), 346–374 (2010)
76. Qin, T., Tao Qin, T.Y.L.: Microsoft Learning to Rank Datasets. `https://www.microsoft.com/en-us/research/project/mslr/` (2010). [Online; accessed 23-Jun-2019]
77. Rani, A.B., Kamal, A.N.B.: Text mining to concept mining: Leads feature location in software system. In: 2018 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pp. 1–7. IEEE (2018)
78. Refaeilzadeh, P., Tang, L., Liu, H.: Cross-Validation. In: Encyclopedia of database systems, pp. 532–538. Springer, Verlag Berlin Heidelberg (2009)
79. Rubin, J., Chechik, M.: A survey of feature location techniques. In: Domain Engineering, pp. 29–58. Springer, Verlag Berlin Heidelberg (2013)
80. Salman, H.E., Seriai, A., Dony, C.: Feature Location in a Collection of Product Variants: Combining Information Retrieval and Hierarchical Clustering. In: The 26th International Conference on Software Engineering and Knowledge Engineering, pp. 426–430 (2014)
81. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval (1986)
82. Shabtai, A., Moskovitch, R., Elovici, Y., Glezer, C.: Detection of Malicious Code by Applying Machine Learning Classifiers on Static Features: A State-of-the-Art Survey. information security technical report **14**(1), 16–29 (2009)
83. Song, Q., Jia, Z., Shepperd, M., Ying, S., Liu, J.: A General Software Defect-Proneness Prediction Framework. IEEE Transactions on Software Engineering **37**(3), 356–370 (2011)
84. Spanoudakis, G., Zisman, A., Pérez-Minana, E., Krause, P.: Rule-Based Generation of Requirements Traceability Relations. Journal of Systems and Software **72**(2), 105–127 (2004)
85. Stikkolorum, D.R., Putten, P.V.D., Sperandio, C., Chaudron, M.: Towards automated grading of uml class diagrams with machine learning. In: BNAIC/BENELEARN (2019)
86. Team, D., et al.: Deeplearning4j: Open-source Distributed Deep Learning for the JVM. Apache Software Foundation License **2** (2016)
87. Vargha, A., Delaney, H.D.: A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics **25**(2), 101–132 (2000)
88. Wang, J., Zhao, P., Hoi, S.C., Jin, R.: Online Feature Selection and its Applications. IEEE Transactions on Knowledge and Data Engineering **26**(3), 698–710 (2014)
89. Wille, D., Holthusen, S., Schulze, S., Schaefer, I.: Interface Variability in Family Model Mining. In: 17th International Software Product Line Conference, pp. 44–51. ACM, New York, USA (2013)
90. Winkler, S., Pilgrim, J.: A Survey of Traceability in Requirements Engineering and Model-Driven Development. Software and Systems Modeling (SoSyM) **9**(4), 529–565 (2010)
91. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Science & Business Media, address = Berlin/Heidelberg, Germany Berlin/Heidelberg, Germany (2012)
92. Zhang, X., Haugen, Ø., Møller-Pedersen, B.: Model Comparison to Synthesize a Model-Driven Software Product Line. In: Proceedings of the 15th International Conference on Software Product Lines, pp. 90–99. IEEE Computer Society, Washington, USA (2011)
93. Zhang, X., Haugen, Ø., Møller-Pedersen, B.: Augmenting Product Lines. In: 19th Asia-Pacific Software Engineering Conference, vol. 1, pp. 766–771. IEEE, New Jersey, US (2012)
94. Zheng, L., Wang, H., Gao, S.: Sentimental Feature Selection for Sentiment Analysis of Chinese Online Reviews. Int. J. Machine Learning & Cybernetics **9**(1), 75–84 (2018)
95. Zhou, Z.H., Feng, J.: Deep Forest: Towards an Alternative to Deep Neural Networks. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, vol. 1, pp. 3553–3559. ijcai.org, Marina del Rey, California (2017)

Ana C. Marcén received the Ph.D. degree in computer science from the Universitat Politècnica de València. She is currently an Associate Professor with the SVIT Research Group, Universidad San Jorge. She publishes her research results and participates in high quality international software engineering conferences and journals, such as the Conceptual Modeling (MoDELS) conference, the Information and Software Technology (IST) journal, and the Journal of Systems and Software (JSS). Her current research interests include model-driven development, feature location, traceability link recovery, and machine learning.

Francisca Pérez is Associate Professor in the SVIT Research Group (https://svit.usj.es) at San Jorge University. She received a PhD in Computer Science from the Polytechnic University of Valencia. Her research interests include Model-Driven Development, Collaborative Information Retrieval, Search-Based Software Engineering, and Variability Modeling. She publishes her research results and participates in high-level international software engineering conferences and journals, such as IEEE Transactions on Software Engineering (TSE), the Automated Software Engineering (AUSE) journal, the Information & Software Technology (IST) journal, and the Journal of Systems and Software (JSS). More about Pérez and her work is available online at http://franciscaperez.com.

Óscar Pastor is currently a Full Professor and the Director of the PROS Research Center, Universitat Politècnica de València, Spain. With a strong background in Conceptual Modeling, Model-driven Development and their practical applications in Information Systems design and development, he is currently leading a multidisciplinary project linking information systems and bioinformatics to designing and implementing tools for conceptual modeling-based interpretation of the Human Genome information.

Carlos Cetina received the Ph.D. degree in computer science from the Polytechnic University of Valencia. He is currently an Associate Professor with Universidad San Jorge and the Head of the SVIT Research Group. His research interests include software product lines and model-driven development. His research results have reshaped software development in world-leader industries from heterogeneous domains ranging from induction hob firmware to train control and management systems. More information about his background can be found at his website: http://carloscetina.com.