Towards Feature Location in Models through a Learning to Rank Approach

Ana C. Marcén, Jaime Font
acmarcen@usj.es
jfont@usj.es
SVIT Research Group
Universidad San Jorge
Autovía A-23 Zaragoza-Huesca
Km.299
Zaragoza, Spain 50830

Óscar Pastor
opastor@pros.upv.es
Centro de Investigación en Mtodos de Producción de Software
Universitat Politècnica de València
Camino de Vera, s/n
Valencia, Spain 46022

Carlos Cetina
ccetina@usj.es
SVIT Research Group
Universidad San Jorge
Autovía A-23 Zaragoza-Huesca
Km.299
Zaragoza, Spain 50830

ABSTRACT

In this work, we propose a feature location approach to discover software artifacts that implement the feature functionality in a model. Given a model and a feature description, model fragments extracted from the model and the feature description are encoded based on a domain ontology. Then, a Learning to Rank algorithm is used to train a classifier that is based on the model fragments and feature description encoded. Finally, the classifier assesses the similarity between a population of model fragments and the target feature being located to find the set of most suitable feature realizations. We have evaluated the approach with an industrial case study, locating features with mean precision and recall values of around 73.75% and 73.31%, respectively (the sanity check obtains less than 35%).

KEYWORDS

Feature Location, Learning to Rank, Model-based development

ACM Reference format:
Ana C. Marcén, Jaime Font, Óscar Pastor, and Carlos Cetina. 2016. Towards Feature Location in Models
through a Learning to Rank Approach. In *Proceedings of 21st International Conference on Software Product Line, Seville, Spain, 25–29 September, 2017 (SPLC'17),* 8 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Feature location is known as the process of finding the set of software artifacts that realize a particular functionality of software system. No maintenance activity can be completed without locating in the first place the software artifact (e.g., code) that is relevant to the specific functionality [10]. Since Feature Location is one of the main activities performed during software evolution [14] and up to an 80% of a system's lifetime is spent on the maintenance and evolution of the system [21], there is a great demand for Feature Location approaches that can help developers to find relevant software artifacts in a family of software products.

Learning to Rank is known as a family of Machine Learning algorithms that automatically address ranking tasks [22]. The topic has gained interest in recent years [9], and Learning to Rank has been applied in a lot of fields [7] like document retrieval, collaborative filtering, expert finding, anti web spam, sentiment analysis, product rating, and feature location.

However, most of the research on Feature Location through Learning to Rank has been directed towards the location of features in source code artifacts [5, 10, 33], neglecting other software artifacts such as models. Therefore, there is a dearth of Feature Location approaches that research how to apply Learning to Rank in order to locate the model elements that realize a feature.

In this work we propose LRFL-M (Learning to Rank for Feature Location in Models), which is an Feature Location approach that locates features in models through Leaning to Rank. The approach is based on Learning to Rank to assess the similarity between a feature description and the model fragments that could be the realizations of this feature. Given feature descriptions and model fragments known beforehand, the LRFL-M approach encodes them based on a domain ontology. Then, the classifier is trained based on the feature descriptions and the model fragments encoded. Finally, the similarity between a population of model fragments and the target feature being located are assessed through the classifier in order to find the set of most suitable feature realizations. Therefore, a rank allows knowing what model fragments best realize the target feature as output.

The presented approach was evaluated in CAF, a worldwide provider of railway solutions. Their trains can be found all over the world in different forms (regular trains, subway, light rail, monorail, etc.). The application of the approach shows that the mean values of precision and recall are 73.75% and 73.31%, respectively, while the sanity check is around 46% less than the presented approach.

The contribution of this paper is twofold. First, we show how to encode model elements and feature descriptions by means of a domain ontology in order to apply Learning to Rank to models. Second, we provide evidence that, with our ontology-based encoding, Learning to Rank is applicable to the problem of feature location in industrial models such as the ones from our industrial partner.

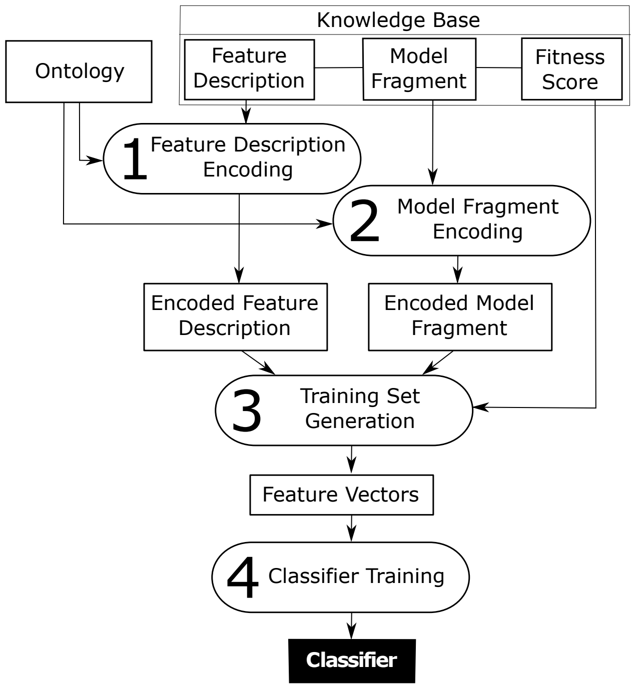


Figure 1: Overview of the Learning to Rank phase of LRFL-M.

The remainder of this paper is structured as follows: Section 2 presents the details of the approach. Section 3 provides the evaluation carried out. Section 4 discusses the approach. Section 5 describes the threats to validity. Finally, Section 6 presents some related work and the paper is concluded with remarks on future work.

2 LEARNING TO RANK FOR FEATURE LOCATION IN MODELS (LRFL-M)

The approach consists of two phases: Learning to Rank and Feature Location. In the first phase, the approach learns how to rank based on a set of feature descriptions and model fragments whose similarity with each other is known beforehand. In the second phase, the approach locates a target feature in a model thanks to the learning obtained in the first phase. As output, some fragments of the model are ranked taking into account their similarity to the target feature.

2.1 The Learning to Rank phase of LRFL-M

In the first phase, a classifier is trained by a Learning to Rank algorithm. Figure 1 shows an overview of the Learning to Rank phase. Rectangular boxes represent the inputs and outputs, while rounded boxes represent the different steps to follow in this phase. Lines indicate that an element is an input or output of one of the steps.

The input consists of the knowledge base and a domain ontology provided by domain experts. Each input is described as follows:

- The **knowledge base** is a set of elements that is generated using the domain experts' experience, documents, and results. Each element is composed of a feature description, a model fragment, and a fitness score. The feature description uses natural language to define the feature that is located in the model fragment. The model fragment consists of an element or a set of elements that belongs to the model. The fitness score determines if the model fragment realizes the feature to a greater or lesser extent. In other words, the fitness score assesses the similarity between the feature description and the model fragment.
- The **domain ontology** represents the main concepts and the relation between them in a specific domain.

The Learning to Rank algorithms train classifiers by using training sets that are composed of feature vectors [6], so neither the feature descriptions nor the model fragments can be understood without encoding them. 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 [19]. On the other hand, in Learning to Rank, a feature is an individual measurable characteristic of the element being observed [8].To avoid misunderstandings, in this article the concept of feature in Learning to Rank is replaced by characteristic. Therefore, each feature vector consists of characteristic-value pairs, and, in the two first steps of this phase, the feature descriptions and the model fragments are encoded into feature vectors, respectively.

(1) *Feature Description Encoding*

In this first step, each feature description is turned into a encoded feature description. First, the main terms of the feature description are extracted using well-established Information Retrieval (IR) techniques: tokenizer, Parts-of-Speech (POS) tagging technique, and stemming techniques [3, 16]. Second, the relevance of these terms is assessed comparing them with the concepts in the domain ontology.

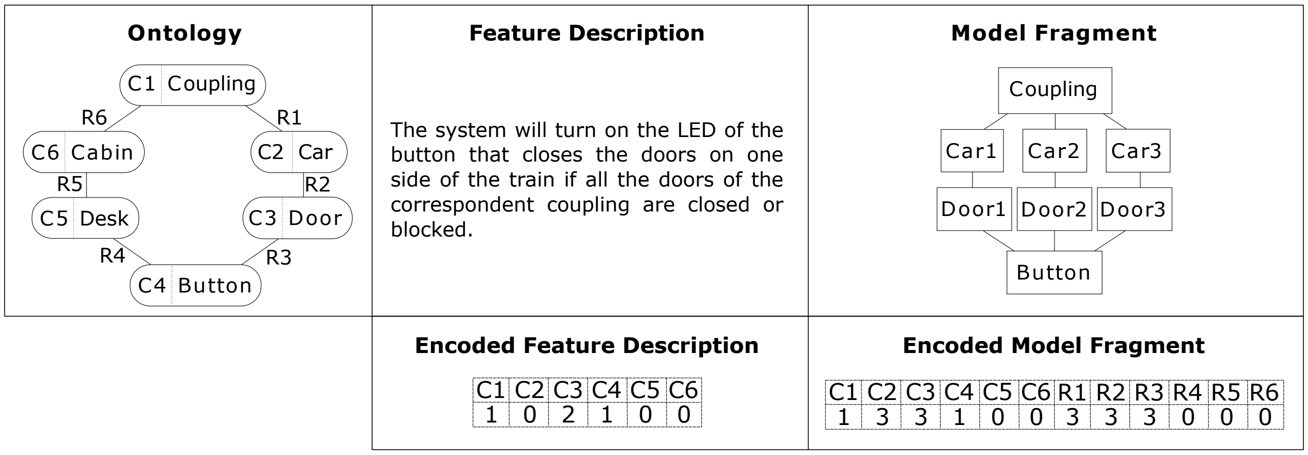
Figure 2 presents an example of the encoding of a feature description and a model fragment based on an ontology. The section on the left shows the ontology provided by a domain expert. In the center, the figure shows a feature description. And, below it, the feature description is encoded as characteristic-value pairs.

Each concept in the ontology is a characteristic in the encoded feature description. Its correspondent value is computed as the frequency of this concept in the feature description. Specifically, the concept is compared against the terms extracted using the IR techniques. This step produces the encoded feature descriptions as output.

(2) *Model Fragment Encoding*

In this second step, the model fragments are encoded based on the domain ontology. However, the encoding is not based only on the concepts of the ontology but also on its relations.

On the one hand, the main terms of the model fragment are extracted taking into account the elements of the model fragment and their properties. Then, the same IR



The system will turn on the LED of the button that closes the doors on one side of the train if all the doors of the correspondent coupling are closed or blocked.

C1	C2	C3	C4	C5	C6
1	0	2	1	0	0

C1	C2	C3	C4	C5	C6	R1	R2	R3	R4	R5	R6
1	3	3	1	0	0	3	3	3	0	0	0

Figure 2: Example of the encoding of a feature description and a model fragment based on an ontology.

techniques (tokenizer, POS tagging technique, and stemming techniques) are applied on the extracted terms. Then, characteristic-value pairs are generated using the concepts of the ontology. Each concept corresponds to a characteristic. Its value is the frequency of the concept in the terms that are extracted from the model fragment.

On the other hand, the relations available in the model fragments are also encoded as characteristic-value pairs. Each relation of the ontology corresponds to a characteristic. Its value is the frequency of the relation in the model fragment. Specifically, this frequency is computed automatically taking into account how the metamodel implements the relations defined in the ontology.

The section on the right of Figure 2 shows a model fragment. On the bottom-right of this figure (Encoded Model Fragment), the left column represents the characteristic-value pairs for the concepts and the right column represents the characteristic-value pairs for the relations. The output of this step is the encoded model fragment, which is composed of the characteristic-value pairs for the concepts and for the relations.

(3) *Training Set Generation*

This step generates the training set that is used to train the classifier. Learning to Rank algorithms use two different sets: one is to train the classifier which is known as the training set; the other one is to perform the ranking by using the classifier, which is known as the test set. Both of them are composed of feature vectors.

In this case, each feature vector is composed of an encoded feature description, an encoded model fragment, and a fitness score. The encoded feature description comes from a feature description that is encoded in the first step. The encoded model fragment comes from a model fragment that is encoded in the second step. The fitness score is the value assigned in the knowledge base to determine the similarity between that feature description and that model fragment.

Following the example of Figure 2, the feature vector would be composed of eighteen characteristic-value pairs and a fitness score. The first six characteristic-value pairs would belong to the encoded feature description. Then, the following twelve characteristic-value pairs would belong to the encoded model fragment. The fitness score would be the correspondent numerical value of the knowledge base.

(4) *Classifier Training*

In this step, the classifier is trained by a Learning to Rank algorithm using the training set that was defined in the previous step. The Learning to Rank algorithm compares the feature vectors of the training set by assessing of the similarity between the fitness scores, the encoded feature descriptions, and the encoded model fragments. Then, the constraints extracted from these comparisons are used by the algorithm to generate the classifier. This classifier is the output of the Learning to Rank phase.

2.2 The Feature Location phase of LRFL-M

Figure 3 depicts an overview of the Feature Location phase. The input consists of the same domain ontology that was used in the previous phase, the target feature description, and the population of model fragments where this target feature is going to be located. First, the target feature description and the model fragments are encoded by using the same techniques described in the first and second steps of the Learning to Rank phase. Then, the test set is generated by using the encodings. Each feature vector of the test set consists of the same encoded feature description and one of the encoded model fragments. Then, the classifier assesses the fitness scores for each feature vector in the the test set. These fitness scores are the output of the Feature Location phase. They allow the model fragments to be ranked according to their similarity to the target feature description.

In the following sections, we describe the case study that we designed to address the evaluation of the approach, as well as its results.

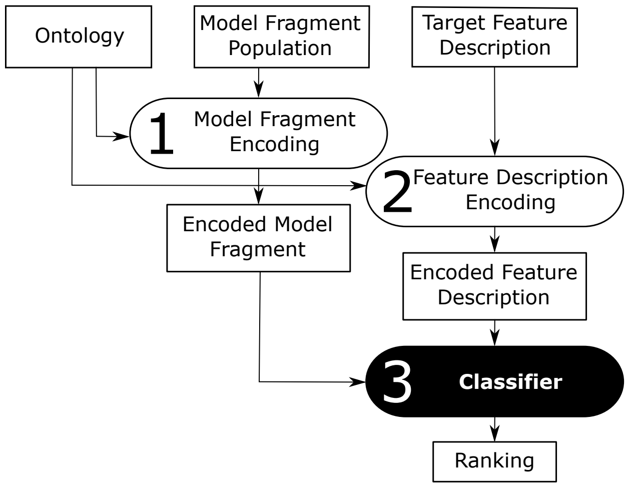

Ontology
Model Fragment Population
Target Feature Description
1 Model Fragment Encoding
2 Feature Description Encoding
Encoded Model Fragment
Encoded Feature Description
3 Classifier
Ranking

Figure 3: Overview of the Feature Location phase of LRFL-M.

3 EVALUATION

This section presents the evaluation that was performed to determine if the presented approach can be used to locate features. The following subsections describe the experimental setup, the case study where we applied the approach, and the results obtained.

3.1 Experimental Setup

Figure 4 shows an overview of the process that was followed to evaluate the approach to locate features in the industrial case study. The top part shows the ontology, the knowledge base, and the oracle for the case study. The **ontology** represents the main concepts and the relations with each other in a specific domain. This ontology was defined by a domain expert from the main concepts in the domain and how they are related. The **knowledge base** is a set of feature descriptions, models fragments that are possible realizations for that feature description, and the fitness score assigned to that model fragment. This knowledge base was constructed by engineers from our industrial partner and then the fitness scores were assigned by a domain expert. The **oracle** consists of a set of features whose traceability between their feature descriptions and model fragments is documented by our industrial partner. The oracle will be considered the ground truth and will be used to evaluate the solutions provided in terms of precision, recall, and the F-measure.

Then, the knowledge base is divided into two different sets. The first one will be encoded to generate the training set in the Learning to Rank phase of the LRFL-M approach. The second one will be encoded to generate the test set in the Feature Location phase of the LRFL-M approach. Therefore, the knowledge base for training is used to generate a classifier following the steps described in the Learning to Rank phase of the LRFL-M. And, in the Feature Location phase of LRFL-M, the knowledge base for testing is fed as input for this classifier. In addition, the Knowledge base for testing is also fed as input for a random classifier (Sanity Check). As a result, we obtained a solution in the form of a model fragment for each classifier. Finally, we computed the precision, recall and F-measure values for each of these solutions.

Precision measures the number of elements from the solution that are correct according to the ground truth (the oracle) and is defined as follows:

$$Precision = \frac{SolutionElements \cap OracleElements}{SolutionElements} \tag{1}$$

Recall measures the number of elements of the solution that are retrieved by the proposed solution and is defined as follows:

$$Recall = \frac{SolutionElements \cap OracleElements}{OracleElements} \tag{2}$$

Finally, F-measure corresponds to the harmonic mean of precision and recall and is defined as follows:

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

The presented approach uses the Eclipse Modeling Framework (EMF) to manipulate the models from our industrial partners and the Common Variability Language (CVL) [29] to manage the fragments of models. The IR techniques that are used to process the language were implemented using OpenNLP [1] for the POS-Tagger and Snowball [2] for the stemming. Finally, SVM-Rank is the algorithm that is used to generate the classifier [18]. SVM-Rank is a well-known Learning to Rank algorithm that is based on the Support Vector Machine (SVM) [26].

3.2 CAF case study

First, we extracted an **oracle** from our industrial partner models. Fy, we obtained four different product models of real world trains, each one of which is composed of around 1200 elements on average. The product models are built using 121 different features that can be present in each product model. Besides the product models, we also extracted the formalization of the variability, which maps each feature to the model fragments that realizes the feature (so we can use it as an oracle).

To create the **knowledge base**, we selected one feature from each of the trains and asked 19 different engineers from our industrial partner to create a model fragment that realizes the feature. Then, each model fragment was assigned a score by a domain expert to determine its correctness.

Using the knowledge base, we performed four test cases, one for each of the features present in the knowledge base. Each test case ranked the model fragments that realized one of the features. The model fragments that realized the other features were used to train the classifier. Table 1 shows how the knowledge base is divided in each test case taking into account one feature for testing and using the other features for training.

This table also shows the number of elements that the knowledge base contains for each test case. These elements correspond to concepts and relations of the ontology, which have to be present so that the feature is realized properly. Therefore, the number of elements depends of the concepts and relations that the feature contains.

Then, for each test case, we followed the experimental setup described in Figure 4. The LRFL-M approach followed the two phases defined above. In the Learning to Rank phase, the knowledge base

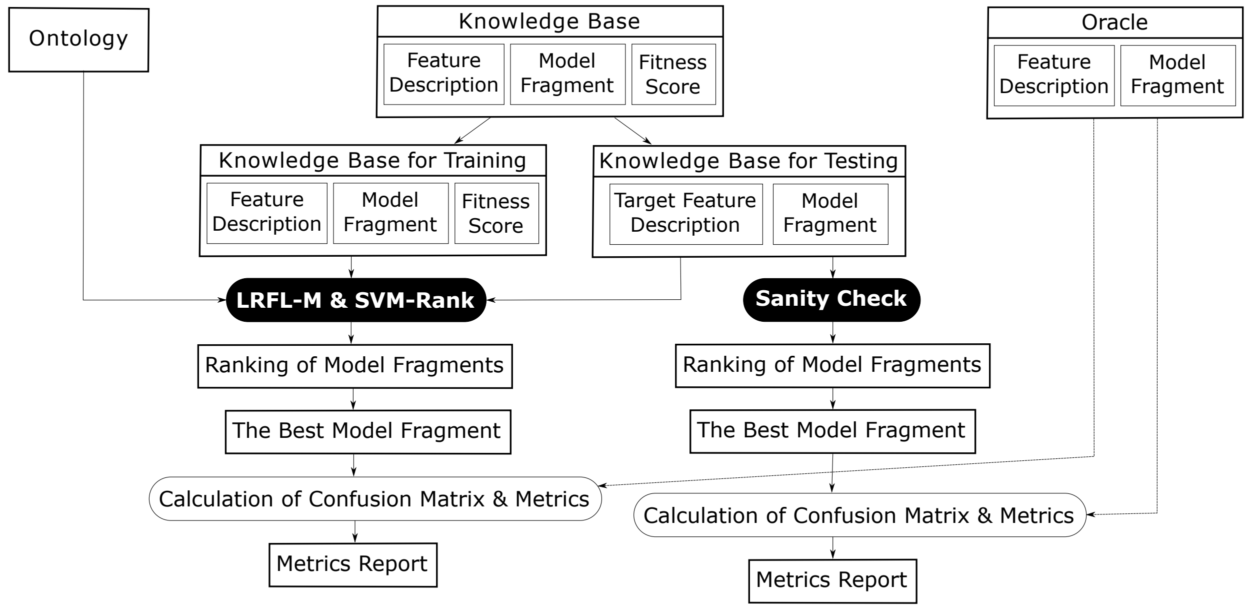


Figure 4: Experimental Setup

Table 1: Division of the knowledge base for each test case (TC) taking into account the features (F), and the number of elements that these features contain.

	Knowledge Base for Training		Knowledge Base for Testing	
	Features	#Elements	Features	# Elements
TC1	F2, F3, F4	14	F1	1
TC2	F1, F3, F4	8	F2	7
TC3	F1, F2, F4	12	F3	3
TC4	F1, F2, F3	11	F4	4

for training was encoded taking into account the ontology and the format required by the Learning to Rank algorithm. The **ontology** defined for this specific domain is composed of 12 concepts and 17 relations for these concepts. Therefore, the encoded feature description was composed of 12 characteristic-value pairs, and the encoded model fragment was composed of 29 characteristic-value pairs.

Moreover, SVM-Rank requires a specific format for its feature vectors [18]. First, each feature vector contains a numerical value which corresponds to our fitness score. Then, it has a numerical value to identify what feature vectors are related to each other. In our case, this value represents the feature that is realized in the feature vector. Therefore, here all the feature vectors that realize the same feature will have the same identifier. Then, the feature vector contains characteristic-value pairs, so the encoded feature description and the encoded model fragment are included after the identifier. Finally, it is possible to add a comment, which was not relevant for the SVM-Rank approach but that may help us to clarify the understanding of the feature vector.

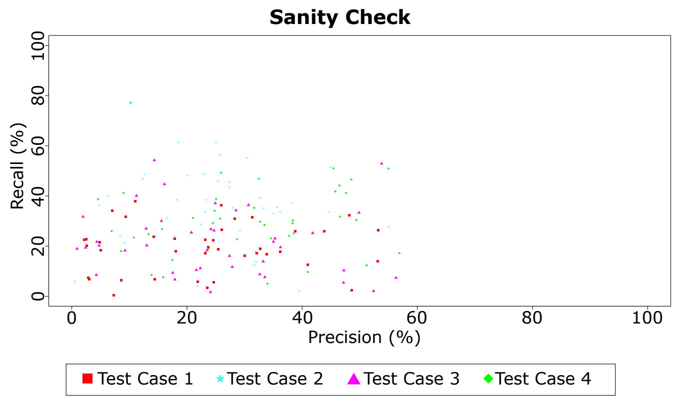
Then, the training set generated was used to train the classifier. In the Feature Location phase of LRFL-M, the knowledge base for testing was encoded taking into account the ontology and the format required by the Learning to Rank algorithm. Then, the testing set was ranked by the classifier that was generated in the previous phase. As a result of the LRFL-M approach, we obtained a ranking of model fragments, and the results were assessed by comparing them to the oracle.

Therefore, each test case returned two results: one for the SVM-Rank classifier, and one for the Sanity Check. Finally, each test case was run 30 times. As suggested by [4], given the stochastic nature of the LRFL-M approach, several repetitions are needed to obtain reliable results.

3.3 Results

This subsection presents the results obtained for each of the tested test cases. The right-hand graph on Figure 5 shows the mean values of precision and recall achieved for each test case when locating the features from the CAF case study using the presented approach. Each point in the chart represents the mean value (for the 30 independent executions) of the two performance indicators (precision on the x axis and recall on the y axis) for one of the test case executions. Similarly, the left-hand graph on Figure 5 shows the mean values of precision and recall achieved for each test case when locating the features from the CAF case study using the Sanity Check.

In addition, in each of the graphs, the points belonging to each of the four test cases are displayed using different symbols. In order words, red squares represent the executions of TC1; blue asterisks represent the executions of TC2; pink triangles represent the executions of TC3; and green diamonds represent the executions of TC4.



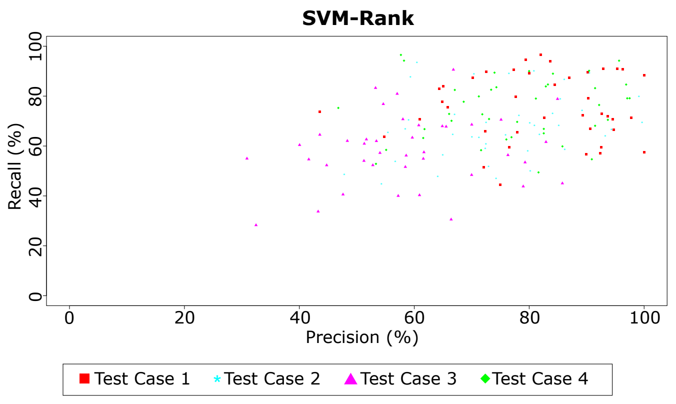


Figure 5: Mean Precision and Recall for the four test cases and both rankings: Sanity Check and SVM-ranking

Table 2: Mean Values and Standard Deviations for Precision, Recall, and F-Measure for SVM and Sanity Check

TC	Measurement	SVM-Ranking	Sanity Check
TC1	Precision ± (σ)	84.06 ± 9.86	23.00 ± 14.95
	Recall ± (σ)	84.12 ± 11.09	19.09 ± 9.77
	F-Measure ± (σ)	83.54 ± 8.25	17.42 ± 10.17
TC2	Precision ± (σ)	69.31 ± 14.93	24.91 ± 11.73
	Recall ± (σ)	69.95 ± 15.40	34.13 ± 14.84
	F-Measure ± (σ)	67.84 ± 11.70	25.78 ± 10.87
TC3	Precision ± (σ)	62.86 ± 17.14	24.93 ± 15.25
	Recall ± (σ)	62.25 ± 13.67	21.33 ± 13.11
	F-Measure ± (σ)	60.15 ± 10.60	18.21 ± 10.94
TC4	Precision ± (σ)	78.79 ± 13.80	30.18 ± 14.75
	Recall ± (σ)	76.94 ± 12.13	30.28 ± 14.39
	F-Measure ± (σ)	77.00 ± 10.01	27.06 ± 12.04
Mean	Precision ± (σ)	73.75 ± 16.28	25.76 ± 14.35
	Recall ± (σ)	73.31 ± 15.38	26.21 ± 14.46
	F-Measure ± (σ)	72.13 ± 13.48	22.11 ± 11.75

In Table 2, we outline the results, which are aggregated for each classifier and test case. We also show the F-measure. The SVM-Rank classifier achieves the best results for all the performance indicators across the four test cases, providing a mean precision value of 73.75%, a recall value of 73.31%, and a combined F-measure of 72.13%. In contrast, the Sanity Check achieves mean values of around 26%, which is approximately 46% less than the presented approach.

4 DISCUSSION

The results reveal that by using our ontology-based encoding, Learning to Rank can be applied to industrial models such as the models of CAF. In the following subsections, we discuss some limitations and the generalization.

4.1 Limitations of our ontology-based encoding

The presented approach relies on sets of information that are divided into three components: the feature description, the model fragment, and the fitness score (which assesses the similarity between the first two components). Specifically, the fitness score is given by an expert in the domain, and it will have an impact on the ranking classifier that is produced by the Learning to Rank algorithm. Therefore, it must be carefully assigned by the domain expert.

Based on the results, the TC1 obtains the best results because the knowledge base for testing is the simplest testing set, as the table 1 shows. While the knowledge base for testing contains only one element, the knowledge base for training contains 14 elements. Therefore, the greater number of elements contained in the knowledge for training helps to better rank the knowledge base for testing. In contrast, the TC3 obtains the worst results because the F3 does not contain any element in common with the other features. While the F1, F2, and F4 have some elements in common, the elements in the F3 are not present in the other features. Therefore, ranking this feature is more complex than to rank the other features. For these reasons, one of the future works will consist in increasing the knowledge base.

In addition, our encoding is based on the presence or absence of concepts and relations from the domain ontology in the feature description and in the model fragment. Although the classifier achieves satisfactory results in our evaluation, it is not enough to capture specific details. For example, two different feature descriptions could contain the same relations and concepts: the first one indicates that the doors of the train will be opened when a button is pressed and the train is stopped, while the other one indicates that the button will be inhibited when the doors are closed and the train is in motion. The two feature descriptions have the same concepts and relations, but they have a different meaning and the correspondent model fragments also have noticeable differences. Therefore, our encoding steps could be improved by taking into account a more complex ontology. In fact, this constitutes our future work.

Moreover, the concepts and the relations defined in the ontology do not have the same relevance to the encoding. In fact, we have observed that some of them include a small deviation when the similarity is computed by the classifier. For example, if a feature description indicates that the state of the pantograph changes when a button is pressed, two model fragments can realize this feature correctly but using different elements. Both of the model fragments would contain the same basic elements (e.g, a button or a pantograph) to function properly. However, one of the model fragments could also have a desk where the button is installed. In this case, the desk is irrelevant to locate the feature. However, it would be included automatically in the encoding thereby adding ambiguity because both model fragments would have the same fitness score and different elements. To solve this, we could use Feature Selection techniques that are widely extended in mining and machine learning applications to simplify the classifiers and to reduce overfitting [13].

Another way to improve the performance of the classifier is by adjusting the parameters of the Learning to Rank algorithm. This approach has been executed using the default parameters [17], but they could be configured to try to improve the results.

4.2 Generalization

The presented approach has been designed to be applied in a specific domain, taking advantage of the experience and knowledge about the domain. An expert in the domain designed the ontology and assigned the fitness scores of the knowledge base. The approach could be applied to any domain, but there must be an ontology and a domain expert to assign the fitness scores for the training.

5 THREATS TO VALIDITY

In this section, we use the classification of threats of validity of [27, 32] to acknowledge the limitations of our approach.

Construct validity: This aspect of validity reflects the extent to which the operational measures that are studied represent what the researchers have in mind. To minimize this risk, our evaluation is performed using three measures: precision, recall, and the F-measure. These measures are widely accepted in the software engineering research community [28].

Internal Validity: This aspect of validity is of concern when causal relations are examined. There is a risk that the factor being investigated may be affected by other neglected factors. The number of model fragments in the knowledge base may look small, but SVM-Rank performs better with small training sets [18]. Therefore, SVM-Rank was the Learning to Rank algorithm selected to reduce this threat.

External Validity: This aspect of validity is concerned with to what extent it is possible to generalize the finding, and to what extent the findings are of relevance for other cases. The LRFL-M approach was designed to locate features in models, but there must be an ontology and a domain expert to assign the fitness scores for the training. If these conditions are satisfied, the features of any domain could be located in models using this approach. Nonetheless, LRFL-M should be applied to other domains before assuring its generalization.

Reliability: This aspect is concerned with to what extent the data and the analysis are dependent on the specific researchers. To reduce this threat, the creation of the ontology and the assignment of the fitness scores were performed by a domain expert who was not involved in the research. Moreover, the feature descriptions and the model fragments were provided by our industrial partner.

6 RELATED WORK

In this section, we present some related works, which are divided into two parts. First, we overview some research papers on Feature Location. Second, we discuss other publications that focus on Feature Location in Models.

6.1 Feature Location Approaches

Typechef [20] provides an infrastructure to locate the code that is associated to a given feature by means of analyzing the #ifdef directives. Trace analysis [11] is a run-time technique that is used to locate features. When the technique is executed, it produces traces that indicate which parts of code have been executed. Some approaches that are related to feature location use LSI to extract the code associated to a feature [21, 24]. These techniques have generally been applied to search for the code of a feature in a given individual product. In contrast, our approach searches for model fragments that implement a feature by means of an ontology-based encoding that enables the application of Learning to Rank algorithms to models.

Some works rely on Learning to Rank techniques to locate features in the code [5, 33]. Tien-Duy et al. focus on Learning to Rank through feature vectors that are based on likely invariants. Xin et al. focus on the terms that are defined in a vocabulary to build the feature vectors. In our approach, we also take advantage of the knowledge of domain experts to define the feature vectors that are based on the ontology created by them. Our approach also performs feature location through Learning to Rank algorithms. However, our approach locates the features in models instead of in code.

Some works rely on ontologies to locate features in code. In [31], a systematic approach is used to locate features by using ontology fragments. Hayashi et al. [15] propose an ontology-based technique to locate features that are defined by natural language sentences. Ratiu et al. [25] present a framework to recover the mappings between entities from an ontology and program elements. Petrenko et al. [23] perform a study about the performance of programmers when they locate features by using ontology fragments. In contrast, our approach locates features in models and the ontology is used to encode both feature descriptions and model fragments.

6.2 Feature Location in Models

Other works focus on the location of features in models using comparisons among models in a family of models [30, 34, 35]. Zhang et al. [34] propose a generic approach to locate the feature realizations by exploring the commonality and the variability of models through their automatic comparison. In [35], the approach is refined to reduce the manual effort required in the formalization of the feature realizations when new product models are included in a product line. In the approach of [30], the variability between models is determined through an exchangeable metric, taking into account different attributes of the models.

However, all of these approaches are based on the location of features through comparisons among the models. In contrast, our approach is applied to a single product model; it relies on a classifier that compares the target feature description with the model.

Font et al. [12] propose a generic approach to locate features in a single model through the use of a genetic algorithm. First, their approach clusters the model fragments into feature realization candidates through Formal Concept Analysis (FCA). Then, Latent Semantic Analysis (LSA) is used to rank the feature realization candidates based on the similarity with the feature description. In contrast to our approach, which benefits from legacy products to train the classifier, in [12] no matter how many legacy products there are, their techniques do not benefit from them.

7 CONCLUSION AND FUTURE WORKS

As part of this work, we have presented a Feature Location approach that targets model fragments as the feature realization artifacts using Learning to Rank. We propose a novel encoding of feature descriptions and model fragments based on an ontology. We propose the use of a Learning to Rank algorithm to learn how to assign fitness scores based on the similarity between feature descriptions and model fragments. As a result, the features located using this approach shows recall rand precision measures of around 73%, while the sanity check remains below 46%. Taking into account these results, our next steps involve the increase of our knowledge base to improve the performance of the classifier, and the enrichment of our ontology to capture specific details to improve our encoding.

ACKNOWLEDGMENTS

This work has been developed with the financial support of the Spanish Ministry of Economy and Competitiveness under the project TIN2016-80811-P and co-financed with ERDF. We also thank the ITEA3 15010 REVaMP^2 Project.

REFERENCES

[1] 2016. Apache OpenNLP: Toolkit for the processing of natural language text. https://opennlp.apache.org/. (2016). [Online; accessed 7-April-2016].

[2] 2016. Snowball : Snowball is a small string processing language designed for creating stemming algorithms for use in Information Retrieval. http://snowball.tartarus.org/. (2016). [Online; accessed 7-April-2016].

[3] Vander Alves, Christa Schwanninger, Luciano Barbosa, Awais Rashid, Peter Sawyer, Paul Rayson, Christoph Pohl, and Andreas Rummler. 2008. An exploratory study of information retrieval techniques in domain analysis. In *Software Product Line Conference, 2008. SPLC'08. 12th International*. IEEE, 67–76.

[4] Andrea Arcuri and Gordon Fraser. 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empirical Software Engineering* 18, 3 (2013), 594–623.

[5] Tien-Duy B Le, David Lo, Claire Le Goues, and Lars Grunske. 2016. A learning-to-rank based fault localization approach using likely invariants. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 177–188.

[6] Gautam Biswas, Jerry B Weinberg, and Douglas H Fisher. 1998. ITERATE: A conceptual clustering algorithm for data mining. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 28, 2 (1998), 219–230.

[7] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*. ACM, 129–136.

[8] Girish Chandrashekar and Ferat Sahin. 2014. A survey on feature selection methods. *Computers & Electrical Engineering* 40, 1 (2014), 16–28.

[9] Olivier Chapelle and Yi Chang. 2011. Yahoo! Learning to Rank Challenge Overview.. In *Yahoo! Learning to Rank Challenge*. 1–24.

[10] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2013. Feature location in source code: a taxonomy and survey. *Journal of software: Evolution and Process* 25, 1 (2013), 53–95.

[11] Andrew David Eisenberg and Kris De Volder. 2005. Dynamic feature traces: Finding features in unfamiliar code. In *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 337–346.

[12] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. 2016. 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*. ACM, 272–282.

[13] Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *Journal of machine learning research* 3, Mar (2003), 1157–1182.

[14] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. 2013. Automatic query reformulations for text retrieval in software engineering. In *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 842–851.

[15] S. Hayashi, T. Yoshikawa, and M. Saeki. 2010. Sentence-to-Code Traceability Recovery with Domain Ontologies. In *2010 Asia Pacific Software Engineering Conference*. 385–394.

[16] Anette Hulth. 2003. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*. Association for Computational Linguistics, 216–223.

[17] Thorsten Joachims. 1999. Svmlight: Support vector machine. *SVM-Light Support Vector Machine http://svmlight. joachims. org/, University of Dortmund* 19, 4 (1999).

[18] Thorsten Joachims. 2009. Svm-rank: Support vector machine for ranking. (2009).

[19] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. DTIC Document.

[20] Christian Kästner, Paolo G Giarrusso, Tillmann Rendel, Sebastian Erdweg, Klaus Ostermann, and Thorsten Berger. 2011. Variability-aware parsing in the presence of lexical macros and conditional compilation. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 805–824.

[21] Dapeng Liu, Andrian Marcus, Denys Poshyvanyk, and Vaclav Rajlich. 2007. 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*. ACM, 234–243.

[22] Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval* 3, 3 (2009), 225–331.

[23] M. Petrenko, V. Rajlich, and R. Vanciu. 2008. Partial Domain Comprehension in Software Evolution and Maintenance. In *2008 16th IEEE International Conference on Program Comprehension*. 13–22.

[24] Denys Poshyvanyk, Yann-Gael Gueheneuc, Andrian Marcus, Giuliano Antoniol, and Vaclav Rajlich. 2007. Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval. *IEEE Transactions on Software Engineering* 33, 6 (June 2007), 420–432.

[25] Daniel Ratiu and Florian Deissenboeck. 2007. From reality to programs and (not quite) back again. In *Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on*. IEEE, 91–102.

[26] Marco Tulio Ribeiro, Nivio Ziviani, Edleno Silva De Moura, Itamar Hata, Anisio Lacerda, and Adriano Veloso. 2015. Multiobjective pareto-efficient approaches for recommender systems. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 4 (2015), 53.

[27] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131–164.

[28] Gerard Salton and Michael J. McGill. 1986. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA.

[29] Andreas Svendsen, Xiaorui Zhang, Roy Lind-Tviberg, Franck Fleurey, Øystein Haugen, Birger Møller-Pedersen, and Gøran K Olsen. 2010. Developing a software product line for train control: A case study of cvl. In *International Conference on Software Product Lines*. Springer, 106–120.

[30] David Wille, Sönke Holthusen, Sandro Schulze, and Ina Schaefer. 2013. Interface variability in family model mining. In *Proceedings of the 17th International Software Product Line Conference co-located workshops*. ACM, 44–51.

[31] L. A. Wilson. 2010. Using ontology fragments in concept location. In *2010 IEEE International Conference on Software Maintenance*. 1–2.

[32] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[33] Xin Ye, Razvan Bunescu, and Chang Liu. 2014. Learning to rank relevant files for bug reports using domain knowledge. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 689–699.

[34] Xiaorui Zhang, Øystein Haugen, and Birger Moller-Pedersen. 2011. Model comparison to synthesize a model-driven software product line. In *Software Product Line Conference (SPLC), 2011 15th International*. IEEE, 90–99.

[35] Xiaorui Zhang, Øystein Haugen, and Birger Møller-Pedersen. 2012. Augmenting product lines. In *2012 19th Asia-Pacific Software Engineering Conference*, Vol. 1. IEEE, 766–771.