Ontological Evolutionary Encoding to Bridge Machine Learning and Conceptual Models: Approach and Industrial Evaluation

Ana C. Marcén, Francisca Pérez, and Carlos Cetina

SVIT Research Group, Universidad San Jorge
Autovía A-23 Zaragoza-Huesca Km.299, 50830, Zaragoza, Spain
`{acmarcen,mfperez,ccetina}@usj.es`

Abstract. In this work, we propose an evolutionary ontological encoding approach to enable Machine Learning techniques to be used to perform Software Engineering tasks in models. The approach is based on a domain ontology to encode a model and on an Evolutionary Algorithm to optimize the encoding. As a result, the encoded model that is returned by the approach can then be used by Machine Learning techniques to perform Software Engineering tasks such as concept location, traceability link retrieval, reuse, impact analysis, etc. We have evaluated the approach with an industrial case study to recover the traceability link between the requirements and the models through a Machine Learning technique (RankBoost). Our results in terms of recall, precision, and the combination of both (F-measure) show that our approach outperforms the baseline (Latent Semantic Indexing). We also performed a statistical analysis to assess the magnitude of the improvement.

Keywords: Machine Learning, Traceability Link Recovery, Evolutionary Computation, Model Driven Engineering

1 Introduction

Machine Learning (ML) is known as the branch of artificial intelligence that gathers statistical, probabilistic, and optimization algorithms, which learn empirically. ML has a wide range of applications, including search engines, medical diagnosis, text and handwriting recognition, image screening, load forecasting, marketing and sales diagnosis, etc. Even though the research on ML has been applied in Software Engineering tasks that target source code artifacts [7, 33], other software artifacts such as conceptual models have been neglected.

Most of the ML techniques are designed to process feature vectors as inputs [8]. Feature vectors are known as the ordered enumeration of features that characterize the object being observed [10]. Therefore, to apply ML techniques in models, the first challenge consists in identifying the features from models and selecting the most suitable ones to encode the models in feature vectors.

In this work, we propose the Ontological Evolutionary Encoding (OnEvEn) approach, which allows models to be encoded in feature vectors. The approach is based on a domain ontology to transform each model to a feature vector and on Evolutionary Computation to perform the selection of the most relevant features. Once the most relevant features have been selected, the approach generates as output the feature vectors from the models according to the selected features. Then the ML techniques can make use of these feature vectors to perform Software Engineering tasks.

The presented approach was evaluated in CAF[1], a worldwide provider of railway solutions. Thanks to our OnEvEn approach, their models were encoded, making it possible for a ML technique (RankBoost [16] that belongs to the family of Learning to Rank) to took advantage of these encoded models to recover the traceability between the requirements and the models. The outcome shows that our approach provides the best results, and proves that the approach can be applied in a real world environment. The statistical analysis of the results assesses the magnitude of the improvement.

The contribution of this paper is twofold. First, we show how to encode models by means of our OnEvEn approach in order to be able to apply ML in models. Second, we provide evidence that by using our OnEvEn approach, ML techniques are applicable to Software Engineering tasks such as traceability link recovery between the requirements and the models.

The remainder of this paper is structured as follows: Section 2 presents our OnEvEn approach. Section 3 provides the evaluation carried out. Section 4 describes the threats to validity. Section 5 presents the related work, and Section 6 concludes the paper.

2 The OnEvEn approach

The objective of the OnEvEn approach is to provide the encoding of a model in the form of a feature vector. To do this, the approach consists of three phases (see Fig. 1): Ontological Encoding, Evolutionary Encoding, and Feature Selection. In the first phase, the approach encodes the model based on a domain ontology. In the second phase, the approach generates a mask, taking advantage of a knowledge base. In the third phase, the approach applies the mask to the feature vector that is the result of the Ontological Encoding. As output, the approach generates a feature vector, which is the encoding of the model.

The input of the approach consists of a model, a domain ontology, and the knowledge base that is provided by domain experts. Specifically, the knowledge base consists in a set of triplets that are generated using the domain experts' experience, results, and documentation. In Fig. 1, each triplet of the knowledge base is composed of a requirement description, a model whose fragment is marked by a dashed square with different background, and an assessment.

The requirement description uses natural language to define the requirement. The model fragment consists of an element or a set of elements that belongs to a model. To formalize these model fragments, we use the Common Variability

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

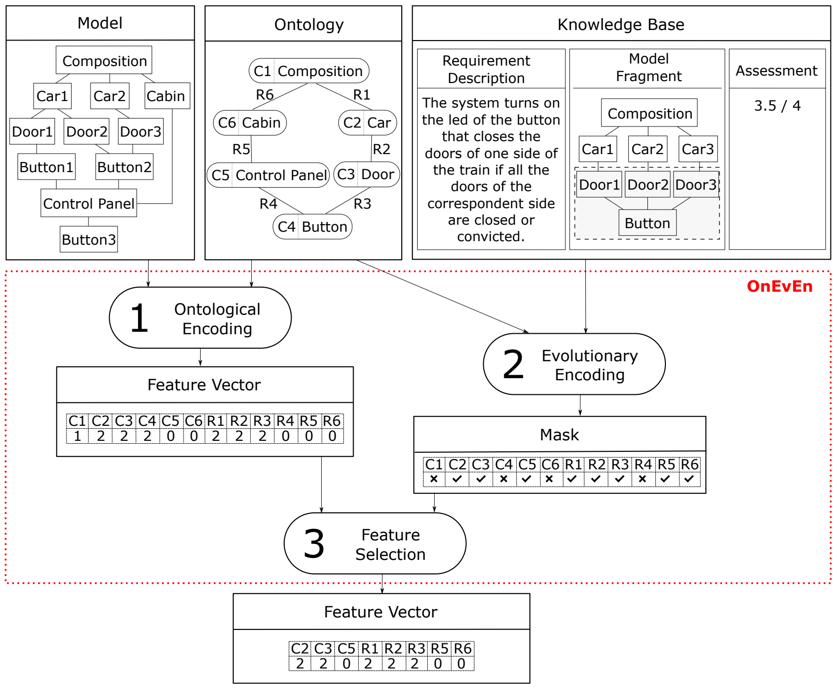

Fig. 1. Overview of our OnEvEn approach.

Language (CVL) [27]. The assessment determines if the model fragment realizes the requirement to a greater or lesser extent. That is, the assessment determines the similarity between the requirement description and the model fragment.

Fig. 1 shows an example of the knowledge base to perform requirement traceability. However, if we plan to perform concept location [21], the knowledge base would be composed of concept descriptions, model fragments, and assessments. Therefore, the knowledge base depends on the Software Engineering task that is going to be performed.

2.1 Ontological Encoding phase

In this first phase, the model is turned into a feature vector based on the domain ontology. We consider each concept and relation in the ontology as a feature in the feature vector. The value of each feature is computed as the frequency of the concept or the relation in the model. Therefore, the output of this phase is a feature vector that represents the model, taking into account the concepts and the relations of the ontology.

The Fig. 1 shows examples of a model, an ontology, and the feature vector that would be generated by this first phase. Concepts and relations of the ontology are features in the feature vector. For example, the concept *Door* is mapped as *C3*, and the relation between the concepts *Cabin-Control Panel* is mapped as R5. Moreover, their values correspond to the number of occurrences of these features in the model. Therefore, the value of the feature *C3* is 3 because there are 3 doors in the model, and the value of the feature *R4* is 1 because there is 1 relation of type *Cabin-Control Panel*.

Once a model is encoded in form of feature vector, ML techniques can use the feature vector to perform Software Engineering tasks. However, the performance of the ML techniques is affected by the redundant and useless features, so Feature Selection is an important step for the approaches that apply ML techniques [23]. For this reason, the following phase performs the selection of the most relevant features.

2.2 Evolutionary Encoding phase

This section details the Evolutionary Encoding phase of OnEvEn approach. This phase involves four steps (see Fig.2): Generation Initial Mask Population, Genetic Operations, Fitness Function, and Top Mask. This phase relies on an Evolutionary Algorithm that iterates a population of masks and evolves them using genetic operations. As output, the phase provides the top mask, which enables only the features that optimize the model encoding.

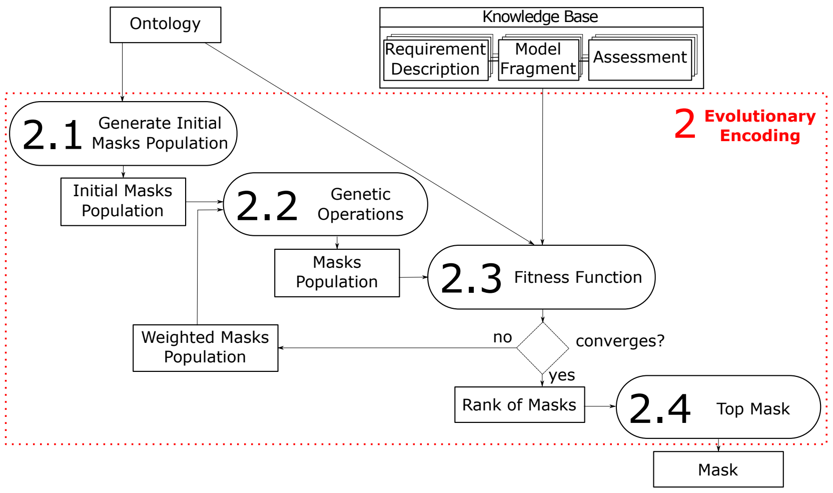

Fig. 2. Details of the Evolutionary Encoding phase of the OnEvEn approach.

Generate Initial Masks Population The first step is to generate randomly a population of masks. Fig.1 shows an example of a mask. Each position of the mask indicates if a concept or relation that belongs to the ontology is enabled o disabled. In order words, if the concept or the relation should be used or not for the encoding.

Genetic Operations The second step is to generate a set of masks that could optimize the model encoding. The generation of masks is done by applying genetic operators that are adapted to work on masks. In other words, new masks that are based on the existing ones are generated through the use of two genetic operators: the mutation and the crossover.

I. The crossover operator is used to imitate the sexual reproduction that is followed by some living beings in nature to breed new individuals. In other words, two individuals mix their genomic information to give birth to a new individual that holds some genetic information from one parent and some from the other one. This could make that the new individual adapt better (or worse) to its living environment depending on the genetic information inherited from its parents. Following this idea, our crossover operator that is applied to masks takes two masks as input and combines them into two new individuals.

II. The mutation operator is used to imitate the mutations that randomly occur in nature when new individuals are born. In other words, a new individual has a small difference with respect to its parents that could make it adapt better (or worse) to its living environment. Following this idea, the mutation operator that is applied to masks takes a mask as input and mutates it into a new one that is produced as output. Specifically, the mutation operator can perform randomly two kinds of modifications based on the features, to enable a feature that is disabled in the mask, or vice versa, to disable a feature that is enabled in the mask.

Fitness Function The third step of the process consists of the assessment of each candidate mask that is produced according to a fitness function. The fitness score of each mask in the population is calculated as follows:

I. *Knowledge Base Encoding* generates a set of feature vectors, which correspond to the triplets of the knowledge base. To encode a triplet, the main terms of the requirement description and the model fragment are extracted using well-established Information Retrieval (IR) techniques: tokenizer, Parts-of-Speech (POS) tagging technique, and stemming techniques. Then, these terms are used to generate the feature vector as the Section 2.1 describes.

II. *Training and Testing* are performed by means of cross-validation [20]. Cross-validation consists of randomly dividing the knowledge base into k-independent partitions. Then, $k-1$ of the partitions are used to train a classifier, which consists in a rule-set that is learnt from a given knowledge base [26]. Then, this classifier is used to test the partition that is left out. This procedure is repeated k times, each time leaving out another partition. This produces k

estimations of the classifier, allowing assessment of its central tendency and variance [18].

III. *Assignment of the Fitness Score* is performed according to the central tendency and variance that are obtained for the classifier. Therefore, the fitness score assesses the relevance of each mask candidate based on how much the results are optimized by using this mask.

IV. *Loop* At this point, if the stop condition is met, the process will stop returning the rank of the masks. If the stop condition has not been met yet, the Evolutionary Algorithm will keep its execution one generation more.

Top Mask The mask with the highest fitness score will be the top mask. This step returns the top mask as output, which allows the model encoding to be optimized by selecting only the most relevant features.

2.3 Feature Selection phase

In the third phase, we apply feature selection on the feature vector that is obtained in the first phase. To do so, the mask that is generated in the second phase is used to reduce the features of the feature vector. As the Fig. 1 shows, each disabled feature in the mask is discarded in the feature vector. Therefore, the feature vector is only composed by the features that are enabled in the mask. As output, our OnEvEn approach returns this feature vector as encoding of the model. In fact, taking into account that the mask is generated to select the most relevant features and avoid the useless and redundant features, the feature vector obtained is able to optimize the performance of ML technique that is used to perform Software Engineering tasks in models.

3 Evaluation

This section presents the evaluation of our approach: the experimental setup, a description of the case study where we applied the evaluation, the implementation details, the obtained results, and the statistical analysis.

3.1 Experimental Setup

The goal of this experiment is to determine if our OnEvEn approach can be used to encode models so that the ML techniques can take advantage of the encoding to perform Software Engineering tasks. In addition, we compare the OnEvEn approach with a baseline.

Fig. 3 shows an overview of the process that was followed to evaluate the baseline and our OnEvEn approach. The top part shows the documentation provided by our industrial partner: the requirements, the product models, and the approved traceability between requirements and product models. The Test Cases are prepared from the documentation provided by our industrial partner, and each test case comprises a requirement and a model fragment of each product model that might be relevant for that requirement. The ontology and the knowledge base that our approach uses as input are provided by a domain expert.

On the one hand, the baseline approach (see the dotted red elements of Fig. 3) uses Latent Semantic Indexing (LSI) to analyze the relevance between requirements provided in Test Cases and the model fragments. There are many Information Retrieval techniques, but most research efforts show better results when applying LSI [24]. On the other hand, our approach (see the solid blue elements of Fig. 3) encodes the models in both the Test Cases and the knowledge base in order to enable the application of ML techniques to models. In this evaluation, we use RankBoost [16] for the ML classifier. RankBoost belongs to the family of Learning to Rank (LETOR) ML algorithms that automatically address ranking tasks [31]. LETOR has been successfully applied in a lot of fields [9] like document retrieval, collaborative filtering, expert finding, anti web spam, sentiment analysis, product rating, and feature location. Our OnEvEn approach enables the application of LETOR to models.

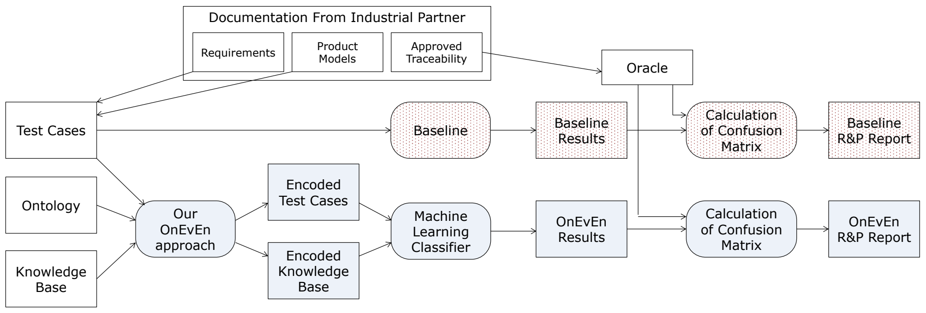

Fig. 3. Experimental Setup

We run the baseline and OnEvEn to obtain as results a ranking of relevant model fragments for each requirement of the Test Cases. Next, we first take the best solution of the ranking of the baseline approach, and then we take the best solution of the ranking of the OnEvEn approach. These best solutions are then compared with an oracle, which is the ground truth. The oracle is prepared using the approved traceability provided by our industrial partner. Once the comparison is performed, a confusion matrix for each approach is calculated.

A confusion matrix is a table that is often used to describe the performance of a classification model (in this case both the baseline and OnEvEn) on a set of test data (the best solutions) for which the true values are known (from the oracle). In our case, each solution outputted by the approaches 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, each model element presence or absence is considered as a classification. The confusion matrix distinguishes between the predicted values and the real values classifying them into four categories:

- True Positive (TP): values that are predicted as true (in the solution) and are true in the real scenario (the oracle).
- False Positive (FP): values that are predicted as true (in the solution) but are false in the real scenario (the oracle).
- True Negative (TN): values that are predicted as false (in the solution) and are false in the real scenario (the oracle).
- False Negative (FN): values that are predicted as false (in the solution) but are true in the real scenario (the oracle).

Then, some performance measurements are derived from the values in the confusion matrix. In particular, we create a report including three performance measurements (recall, precision, and F-measure), for each of the test cases for both the baseline and OnEvEn.

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

$$Recall = \frac{TP}{TP + FN}$$

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

$$Precision = \frac{TP}{TP + FP}$$

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

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2TP + FP + FN}$$

Recall values can range between 0% (which means that no single model element from the realization of the requirement obtained from the oracle is present in any of the model fragments of the solution) to 100% (which means that all the model elements from the oracle are present in the solution). Precision values can range between 0% (which means that no single model fragment from the solution is present in the realization of the requirement obtained from the oracle) to 100% (which means that all the model fragments from the solution are present in the requirement realization from the oracle). A value of 100% precision and 100% recall implies that both the solution and the requirement realization from the oracle are the same.

3.2 CAF case study

The case study where we applied our approach was CAF, a worldwide provider of railway solutions. Their 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 through its vehicles and cabins. These pieces of equipment are often designed and manufactured by different providers, and their aim 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 equipment cooperate to achieve the train functionality, while guaranteeing compliance with the specific regulations of each country.

Our evaluation is made up of 29 test cases, 247 concepts and 161 relationships in the ontology, and 102 triplet in the knowledge base. It is important to highlight that the requirements and the models of the knowledge base are different from the requirements and models of the test cases. The requirements have about 50 words and the models have about 1200 elements. For each test case, we followed the experimental setup described in Fig. 4. Finally, each test case was run 30 times. As suggested by [6], given the stochastic nature of OnEvEn approach, several repetitions are needed to obtain reliable results.

3.3 Implementation details

We have used the Eclipse Modeling Framework to manipulate the models and CVL to manage the model fragments. The IR techniques used to process the language have been implemented using OpenNLP [1] for the POSTagger and the English (Porter 2) [3] stemming algorithm. LSI has been implemented using the Efficient Java Matrix Library (EJML [2]). The genetic operations are built upon the Watchmaker Framework for Evolutionary Computation [13]. Finally, RankBoost has been implemented using the library RankLib [11].

For the settings of the evolutionary algorithm of OnEvEn, we have mainly chosen values that are commonly used in the literature [25]. As suggested by [6], tuned parameters can outperform default values generally, but they are far from optimal in individual problem instances. Therefore, the objective of this paper is not to tune the values to improve the performance of our algorithm.

3.4 Results

This subsection presents the results obtained for each of the Test Cases by both the baseline and OnEvEn. Fig. 4 shows the charts with the recall and precision results for the baseline (in the left side of the figure) and OnEvEn (in the right side of the figure). A dot in the graph represents the average result of precision and recall for each of the 29 Test Cases for the 30 repetitions.

Table 1 shows the mean values of recall, precision and F- measure of the graphs for both the baseline and OnEvEn. OnEvEn obtains the best results in recall and precision, providing an average value of 90.47% in recall and 75.19% in precision. The baseline obtains an average value of 84.22% in recall and 43.97% in precision. Hence, OnEvEn outperforms the baseline.

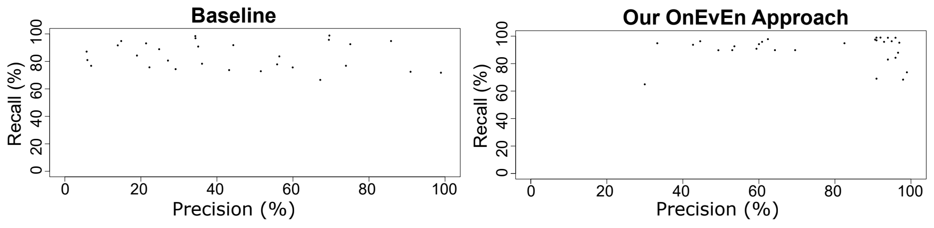

Fig. 4. Mean Recall and Precision values for baseline and OnEvEn approaches

Table 1. Mean Values and Standard Deviations for Precision, Recall, and the F-Measure for Baseline and OnEvEn

	Recall ± (σ)	Precision ± (σ)	F-measure ± (σ)
Baseline	84.22 ± 9.58	43.97 ± 26.81	52.61±23.35
OnEvEn	90.47 ± 9.68	75.19 ± 22.37	79.99±15.33

3.5 Statistical analysis

Statistically significant differences can be obtained even if they are so small as to be of no practical value [5]. Then it is important to assess if an approach 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 use Vargha and Delaney's $\hat{A}_{12}$ [28]. $\hat{A}_{12}$ measures the probability that running one approach yields higher values than running another approach. If the two approaches are equivalent, then $\hat{A}_{12}$ will be 0.5.

The $\hat{A}_{12}$ value for recall between our OnEvEn approach and the baseline is 0.6938, which means that we would obtain better results for recall in 69.38% of the runs with OnEvEn. With regard to the precision, the $\hat{A}_{12}$ value between OnEvEn and the baseline is 0.8056, which shows a superiority of OnEvEn since its results are better in 80.56% of the runs. Hence, these results confirm that the use of our OnEvEn approach has impact on the results, specially on the results for precision.

4 Threats to validity

In this section, we use the classification of threats of validity of [29] 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.

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. RankBoost tend to overfit when the dataset is not large enough and there are many features [30]. Therefore, the number of triplets in our knowledge base may look small. However, Feature Selection in ML enables to avoid overfitting [19] so this threat has been reduced by Feature Selection through the Evolutionary Algorithm.

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. Our OnEvEn approach is designed to encode models for using ML techniques, but there must be an ontology and a knowledge. If these conditions are satisfied, the models of any domain could be encoded using this approach. Nonetheless, OnEvEn 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 knowledge were performed by a domain expert who was not involved in the research. Moreover, the requirements descriptions and the product models were provided by our industrial partner.

5 Related Work

In this section, we present the related works, which are divided into two parts. First, we overview research on Feature Selection. Second, we overview research papers on Requirements Traceability.

5.1 Feature Selection

Haiduc et al. [17] perform feature selection among 21 measures using the gain ratio technique during the retrieval of software artifacts. Ye et al. [34] apply feature selection in mapping bug reports to identify the features that have the most impact on the ranking performance. However, our approach makes use of models instead of other software artifacts such as source code and of a domain ontology as a basis for identifying the features.

Evolutionary Computation techniques have also recently been applied in Feature Selection. Xue et al. [32] present a survey of the state-of-art work on Evolutionary Computation for feature selection in different fields such as image analysis, text mining, and gene analysis. Our approach also takes advantage of Evolutionary Computation to perform feature selection, but it focuses on a domain ontology to encode models and on these encoded models to perform Software Engineering tasks.

5.2 Requirements Traceability

CERBERUS [15] provides a hybrid technique that combines information retrieval, execution tracing, and prune dependency analysis allowing to trace requirements to source code. Eaddy et al. [14] presents a systematic methodology for identifying which code is related to which requirement, and a suite of metrics for quantifying the amount of crosscutting code. Antoniol et al. [4] propose a method based on information retrieval to recover traceability links between source code and free text documents, such as, requirement specifications, design documents, manual pages, system development journals, error logs, and related maintenance reports. Zisman et al. [35] automate the generation of traceability relations between textual requirement artifacts and object models using heuristic rules. These approaches recover the traceability between source code and requirements. In contrast, our work recovers the traceability between requirements and models instead of source code.

Some works rely on models as the software artifacts to perform traceability. De Lucia et al. [12] present a traceability recovery method and tool based on LSI in the context of an artifact management system. Marcus and Maletic [22] use LSI for recovering the traceability relations between source code and documentation (manual, design documentation, requirement documents, test suites, etc.). Our approach makes it possible for a ML technique (Rank Boost) to take advantage of encoded models to recover traceability links between the requirements and the models. Our results show that Rank Boost significantly outperforms LSI in traceability link recovery between the requirements and the models of our industrial partner.

6 Conclusion

Machine Learning (ML) has a wide range of successful applications but current research efforts have neglected the application of ML to models. In this paper, we propose OnEvEn approach that encodes models in order to enable the application of ML techniques to models. We also show that by using our OnEvEn approach, ML techniques are applicable to Software Engineering tasks such as traceability link recovery between the requirements and the models.

We evaluate our OnEvEn approach in terms of precision, recall and F-measure. To do so, we compared it to a baseline in an industrial domain (firmware of train PLCS with CAF). We report our evaluation, including: experimental setup, results, statistical analysis, and threats to validity.

The results show that enabling the application of ML techniques by means of OnEvEn pays off for traceability link recovery. Results also show that our approach can be applied in real world environments. The statistical analysis of the results assesses the magnitude of the improvement of our approach.

Acknowledgments. This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the project Model-Driven Variability Extraction for Software Product Line Adoption (TIN2015-64397-R). We also thank DataMe (TIN2016-80811-P) from MINECO and IDEO (PROMETEOII/2014/039) from the Generalitat Valenciana.

References

1. Apache opennlp: Toolkit for the processing of natural language text. https://opennlp.apache.org/, [Online; accessed April-2017]
2. Efficient java matrix library. http://ejml.org/, [Online; accessed April-2017]
3. The english (porter2) stemming algorithm. http://snowball.tartarus.org/algorithms/english/stemmer.html, [Online; accessed April-2017]
4. 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)
5. Arcuri, A., Briand, L.: A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Softw. Test. Verif. Reliab. 24(3), 219–250 (May 2014)
6. 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)
7. 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 (2016)
8. Bianchini, M., Maggini, M., Jain, L.C.: Handbook on neural information processing. Springer (2013)
9. Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H.: Learning to rank: From pairwise approach to listwise approach. In: Proceedings of the 24th International Conference on Machine Learning. pp. 129–136. ICML '07, ACM, New York, NY, USA (2007)
10. Chandrashekar, G., Sahin, F.: A survey on feature selection methods. Computers & Electrical Engineering 40(1), 16–28 (2014)
11. Dang, V.: The lemur project - wiki - ranklib. http://sourceforge.net/p/lemur/wiki/RankLib/ (2013), [Online; accessed April-2017]
12. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Enhancing an artefact management system with traceability recovery features. In: Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on. pp. 306–315. IEEE (2004)
13. Dyer, D.: The watchmaker framework for evolutionary computation (evolutionary/genetic algorithms for java). http://watchmaker.uncommons.org/, [Online; accessed April-2017]
14. Eaddy, M., Aho, A., Murphy, G.C.: Identifying, assigning, and quantifying crosscutting concerns. In: Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques. p. 2 (2007)
15. Eaddy, M., Aho, A.V., Antoniol, G., Guéhéneuc, Y.G.: Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In: ICPC 2008 conference. pp. 53–62. IEEE (2008)
16. 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)
17. Haiduc, S., Bavota, G., Oliveto, R., De Lucia, A., Marcus, A.: Automatic query performance assessment during the retrieval of software artifacts. In: International Conference on Automated Software Engineering. pp. 90–99. ACM (2012)
18. Hirzel, A.H., Le Lay, G., Helfer, V., Randin, C., Guisan, A.: Evaluating the ability of habitat suitability models to predict species presences. ecological modelling 199(2), 142–152 (2006)

19. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. Machine learning: ECML-98 pp. 137–142 (1998)
20. Kohavi, R., et al.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Ijcai. vol. 14, pp. 1137–1145. Stanford, CA (1995)
21. 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 (Nov 2004)
22. Marcus, A., Maletic, J.I.: Recovering documentation-to-source-code traceability links using latent semantic indexing. In: Software Engineering, 2003. Proceedings. 25th International Conference on. pp. 125–135. IEEE (2003)
23. Navot, A., Shpigelman, L., Tishby, N., Vaadia, E.: Nearest neighbor based feature selection for regression and its application to neural activity. Advances in Neural Information Processing Systems 18, 995 (2006)
24. 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 (Jun 2007)
25. Sayyad, A.S., Ingram, J., Menzies, T., Ammar, H.: Scalable product line configuration: A straw to break the camel's back. In: Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on. pp. 465–474 (Nov 2013)
26. 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)
27. Svendsen, A., Zhang, X., Lind-Tviberg, R., Fleurey, F., Haugen, Ø., Møller-Pedersen, B., Olsen, G.K.: Developing a software product line for train control: A case study of cvl. In: International Conference on Software Product Lines. pp. 106–120. Springer (2010)
28. 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)
29. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science & Business Media (2012)
30. Wolf, L., Martin, I.: Robust boosting for learning from few examples. In: Computer Vision and Pattern Recognition, 2005. vol. 1, pp. 359–364. IEEE (2005)
31. Xuan, J., Monperrus, M.: Learning to Combine Multiple Ranking Metrics for Fault Localization. In: Proceedings of the 30th International Conference on Software Maintenance and Evolution (2014)
32. Xue, B., Zhang, M., Browne, W.N., Yao, X.: A survey on evolutionary computation approaches to feature selection. IEEE Transactions on Evolutionary Computation 20(4), 606–626 (2016)
33. Ye, X., Bunescu, R., Liu, C.: 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. pp. 689–699. ACM (2014)
34. Ye, X., Bunescu, R., Liu, C.: Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation. IEEE Transactions on Software Engineering 42(4), 379–402 (2016)
35. Zisman, A., Spanoudakis, G., Pérez-Miñana, E., Krause, P.: Tracing software requirements artifacts. In: Software Engineering Research and Practice. pp. 448–455 (2003)