Automating Traceability Link Recovery through Classification

Chris Mills
Florida State University
Tallahassee, Florida, USA
chris.mills0905@gmail.com

ABSTRACT

Traceability Link Recovery (TLR) is an important software engineering task in which a stakeholder establishes links between related items in two sets of software artifacts. Most existing approaches leverage Information Retrieval (IR) techniques, and formulate the TLR task as a retrieval problem, where pairs of similar artifacts are retrieved and presented to a user. These approaches still require significant human effort, as a stakeholder needs to manually inspect the list of recommendations and decide which ones are true links and which ones are false. In this work, we aim to automate TLR by re-imagining it as a binary classification problem. More specifically, our machine learning classification approach is able to automatically classify each link in the set of all potential links as either valid or invalid, therefore circumventing the substantial human effort required by existing techniques.

CCS CONCEPTS

• **Software and its engineering → Traceability**;

KEYWORDS

Traceability, Machine Learning, Classification

ACM Reference Format:
Chris Mills. 2017. Automating Traceability Link Recovery through Classification. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 3 pages.
https://doi.org/10.1145/3106237.3121280

1 INTRODUCTION

Traceability Link Recovery (TLR) is an important software engineering task in which explicit links are established between related documents in different software artifact sets (e.g., source code, requirements, design documents, use cases, test cases, etc.). Having reliable traceability links between artifacts has been shown to improve the software maintenance process [15, 16]. While the traditional TLR approach involves a stakeholder establishing these links manually, this process is error-prone and time-intensive. To mitigate these challenges, researchers have often formulated TLR as a retrieval problem. That is, given one set of artifacts as queries, use Information Retrieval (IR) techniques to retrieve a list of similar artifacts from the other artifact set, ordered by their textual similarity to the query. While this type of approach can help stakeholders perform TLR by increasing the likelihood of finding true links, it still requires a substantial amount of human effort to investigate the list of results and differentiate the true links from the false ones.

Ideally, the perfect TLR process should be fully automated: provided with two sets of artifacts, a TLR approach should return all valid links between them. In this work we propose an approach that takes significant steps towards this goal and involves re-imagining TLR as a binary classification problem. Using machine learning techniques, our approach automatically classifies each potential link as a true or false one based on a set of features describing the two artifacts in the link and their relationship. The machine learning classifier is trained to predict the validity of each possible link using patterns mined from historical data. The results of our evaluation indicate that our approach is able to correctly classify the majority of valid links, with a low error rate.

2 RELATED WORK

The majority of current approaches designed to assist with TLR use IR techniques, such as probabilistic [3, 4] and vector space models (VSM) [2], Latent Semantic Indexing (LSI) [17], Latent Dirichlet Allocation (LDA) [5] and so on. For a thorough overview of IR for TLR, Borg et al. [6] provide a systematic mapping study of the field. Our work differs from these TLR approaches as it uses a classifier to automatically identify the true and false links, rather than leaving this task for the developer.

Though different from our approach, some existing work has used classification approaches in the context of TLR. Cleland-Huang et. al. [9] first proposed a probabilistic classifier trained on a set of indicator words for non-functional requirements, which was subsequently used for linking regulatory codes with project requirements [8] and architectural tactics to source code [20]. Our work differs from theirs, as it does not require the intermediate step of manually identifying indicator terms, and does not use a pre-determined threshold for classification. Rather, our approach is completely automatic. Falessi et al. [10] applied machine learning to the task of predicting the number of valid links remaining in an IR result list; however, they did not address the task of predicting the identity of remaining valid links as done in this work.

3 THE APPROACH

The proposed approach uses machine learning classifiers to automatically differentiate valid from invalid links using patterns mined from existing, historical traceability data [19]. To represent traceability links within these models, we introduce three types of

Table 1: Software Systems Used in Evaluation

System	Possible Links	Valid Links	Artifact Types
CM-1	1166	45 (3.86%)	High-R, Low-R
eAnci	7645	554 (7.25%)	UC, CC
eTour	6728	366 (5.44%)	UC, CC
SMOS	6700	1044 (15.58%)	UC, CC
iTrust	1551	58 (3.74%)	UC, CC
EasyClinic	1410	93 (6.60%)	UC, CC
EasyClinic	2961	204 (6.89%)	TC, CC
EasyClinic	1260	83 (6.59%)	ID, TC
EasyClinic	600	26 (4.33%)	ID, UC
EasyClinic	1890	63 (3.33%)	TC, UC
EasyClinic	940	69 (7.34%)	ID, CC
Total	**32821**	**2605 (7.94%)**	

High-R = High-level Requirements, Low-R = Low-level Requirements,
UC = Use Cases, CC = Code Classes, ID = Interaction Diagrams,
TC = Test Cases

features: IR Ranking, Query Quality (QQ), and Document Statistics Features:

- **IR Ranking**: These features represent the similarity of the two documents in a possible link as a pair of ranks derived from using either document as a query to find the other. Features can be generated for any number of IR techniques, and in this work we used standard VSM, BM25, and two language smoothing models: Jelinek Mercer and Dirichlet.
- **QQ**: These features help the model differentiate cases in which IR Ranking features are high (i.e., the documents in a possible link have low textual similarity) because the link is invalid from those in which the features are artificially elevated due to one or both documents being poor quality queries. Metrics from [18] are applied to both documents in the potential link.
- **Document Statistics**: These features represent basic statistics about the documents (i.e., software artifacts) themselves, namely the size of each document (i.e., number of non-unique words), the size of each document's vocabulary (i.e., number of unique words), and the percentage of terms that overlap between the two documents.

Using these features, we investigate several classification algorithms: Random Forest, Decision Trees (J48), Näive Bayes, and k-Nearest Neighbors (KNN) with $k = 5$. In each case, the Weka[1] implementation is used with default parameters. Table 1 shows the data used in our evaluation of the approach, which includes eleven datasets from six software projects, involving six different types of artifacts. These datasets were chosen as they have frequently been used to evaluate new techniques for TLR [1, 11, 12]. As expected, there is a significant class imbalance between valid and invalid links, at an average ratio of $1 : 13$. Unless addressed, this can represent an obstacle for the correct classification of valid links. Therefore, we apply Synthetic Minority Oversampling TEchnique (SMOTE) [7] and majority undersampling [13] to balance the data for our evaluation.

When evaluating our approach we are most concerned with two metrics: True Positive Rate (TPR) and False Positive Rate (FPR), which are given by equations 1 and 2 respectively. In both we use the abbreviations TP (true positives), FP (false positives), TN (true negatives), and FN (false negatives) to denote the number of each provided by model evaluations. TPR is equivalent to Recall, and represents the percentage of valid links that were retrieved by the approach. FPR is the proportion of false positives in the classification (i.e., invalid links predicted to be valid links).

$$TPR = \frac{TP}{TP + FN} \tag{1}$$

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

Note that each of the datasets involved in this study are comprised of three components: textual representations of two software artifact sets and a list of known valid links derived from the original developers of the systems. We use standard ten-fold cross-validation to analyze the performance of each aforementioned algorithm. Using this technique, we randomly segment the available data into two sets: the training and testing sets. For each trial, we use 90% of the data to train a model that is then evaluated on the remaining 10%. This process is performed ten times, ensuring that for each trial a single data point is only included in one testing set. The results of these trials are then averaged to obtain a single set of performance metrics.

[1]http://www.cs.waikato.ac.nz/ml/weka/

Table 2: Average TPR and FPR for each classification algorithm using either SMOTE or majority undersampling

	SMOTE		Undersampling	
Algorithm	**TPR**	**FPR**	**TPR**	**FPR**
J48	0.657	0.041	0.870	0.152
KNN	0.656	0.057	0.882	0.276
Näive Bayes	**0.741**	0.194	0.836	0.234
Random Forest	0.695	**0.017**	**0.927**	**0.122**

4 INITIAL RESULTS

Table 2 shows the results from our preliminary evaluation. We aim to determine the "best" classification algorithm and balancing technique to use for TLR. Because the ultimate goal of this research is to develop a fully automated approach, minimizing FPR is of particular interest. This is because even though our aim is to correctly classify a large percentage of the valid links, if these valid links are buried in a large number of false positives, a user would be needed to refute those false positives, therefore invalidating the automatic nature of the approach.

In terms of TPR when using SMOTE, Näive Bayes is able to retrieve the largest percentage of valid links on average; however, Random Forest retrieves less than 5% fewer links with a dramatic decrease in FPR. That is, it retrieves slightly fewer links with much higher reliability. In the case of undersampling, Random Forest is able to retrieve more than 90% of the valid links on average, more than all the other algorithms, while also maintaining a lower FPR. These preliminary results suggest that Random Forest is the most promising classifier for this task. Furthermore, while undersampling improves TPR, it also increases FPR. These differences between balancing approaches suggests that the mechanism used to address class imbalance is important to overall performance, and that some combination of over- and undersampling is needed. While promising, the results also indicate potential for future improvements, discussed in the following section.

5 CONTRIBUTIONS AND FUTURE WORK

The major contribution of this work is a novel approach towards the automation of TLR which reformulates the task as a binary classification problem. Preliminary results show that our approach has promise, as it is able to accurately identify the majority of the true links, achieving a recall of 0.7 at the low FPR of 0.017. Therefore, while complete automation of TLR has yet to be achieved, our proposed approach lays groundwork for significantly reducing the substantial human effort required by current approaches to TLR.

Future work will perform a more rigorous evaluation of the approach by also comparing it to existing IR-based techniques. We will also focus on improving the approach in various ways. First, we will perform more in-depth feature engineering and selection to tune our representations, in an effort to optimize model performance. Second, while the initial evaluation uses algorithms with default parameters, we will utilize more modern approaches that leverage genetic algorithms [21] or machine learning [14] to establish near-optimal parameter configurations. Moreover, additional mechanisms to address the inherent class imbalance problem will be explored. Additionally, because existing IR techniques do not require training data, this approach will be extended to situations where labeled training data is not available by constructing cross-project classification models. Finally, we will investigate using this approach as a pre-processing step that enhances current semi-automatic approaches for high impact situations in which complete automation is not possible or desired.

REFERENCES

[1] Nasir Ali, Yann-Gaël Gueheneuc, and Giuliano Antoniol. 2011. Requirements traceability for object oriented systems by partitioning source code. In *18th Working Conf. on Reverse Engineering (WCRE)*. IEEE, 45–54.

[2] Giuliano Antoniol, Gerardo Canfora, G Casazza, and A De Lucia. 2000. Information retrieval models for recovering traceability links between code and documentation. In *Proceedings. Intl. Conf. on Software Maintenance, 2000.* IEEE, 40–49.

[3] Giuliano Antoniol, Gerardo Canfora, G Casazza, A De Lucia, and Ettore Merlo. 2000. Tracing object-oriented code into functional requirements. In *Proceedings. IWPC 2000. 8th Intl. Workshop on Program Comprehension, 2000.* IEEE, 79–86.

[4] Giuliano Antoniol, Gerardo Canfora, Andrea De Lucia, and Ettore Merlo. 1999. Recovering code to documentation links in OO systems. In *Proceedings. Sixth (1999) Working Conf. on Reverse Engineering*. IEEE, 136–144.

[5] Hazeline U Asuncion, Arthur U Asuncion, and Richard N Taylor. 2010. Software traceability with topic modeling. In *Proceedings of the 32nd ACM/IEEE Intl. Conf. on Software Engineering-Volume 1*. ACM, 95–104.

[6] Markus Borg, Per Runeson, and Anders Ardö. 2014. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering* 19, 6 (2014), 1565–1616.

[7] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[8] Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Emenecker. 2010. A machine learning approach for tracing regulatory codes to product specific requirements. In *Proceedings of the 32nd ACM/IEEE Intl. Conf. on Software Engineering-Volume 1*. ACM, 155–164.

[9] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. 2007. Automated classification of non-functional requirements. *Requirements Engineering* 12, 2 (2007), 103–120.

[10] Davide Falessi, Massimiliano Di Penta, Gerardo Canfora, and Giovanni Cantone. 2016. Estimating the number of remaining links in traceability recovery. *Empirical Software Engineering* (2016), 1–32.

[11] Malcom Gethers, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. 2011. On integrating orthogonal information retrieval methods to improve traceability recovery. In *27th IEEE Intl. Conf. on Software Maintenance (ICSM)*. IEEE, 133–142.

[12] Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, E Ashlee Holbrook, Sravanthi Vadlamudi, and Alain April. 2007. REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering* 3, 3 (2007), 193–202.

[13] Haibo He and Edwardo A. Garcia. 2009. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering* 21, 9 (Sept. 2009), 1263–1284. DOI: http://dx.doi.org/10.1109/TKDE.2008.239

[14] Sugandha Lohar, Sorawit Amornborvornwong, Andrea Zisman, and Jane Cleland-Huang. 2013. Improving trace accuracy through data-driven configuration and composition of tracing features. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 378–388.

[15] Patrick Mader and Alexander Egyed. 2012. Assessing the effect of requirements traceability for software maintenance. In *2012 28th IEEE Intl. Conf. on Software Maintenance (ICSM)*. IEEE, 171–180.

[16] Patrick Mäder and Alexander Egyed. 2015. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering* 20, 2 (2015), 413–441.

[17] Andrian Marcus, Jonathan Maletic, and Andrey Sergeyev. 2005. Recovery of traceability links between software documentation and source code. *Intl. Journal of Software Engineering and Knowledge Engineering* 15, 05 (2005), 811–836.

[18] Chris Mills, Gabriele Bavota, Sonia Haiduc, Rocco Oliveto, Andrian Marcus, and Andrea De Lucia. 2017. Predicting Query Quality for Applications of Text Retrieval to Software Engineering Tasks. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 26, 1 (2017), 3.

[19] Chris Mills and Sonia Haiduc. 2017. A Machine Learning Approach for Determining the Validity of Traceability Links. In *Proceedings of the 39th Intl. Conf. on Software Engineering Companion (ICSE-C '17)*. IEEE Press, Piscataway, NJ, USA, 121–123. DOI: http://dx.doi.org/10.1109/ICSE-C.2017.86

[20] Mehdi Mirakhorli, Yonghee Shin, Jane Cleland-Huang, and Murat Cinar. 2012. A tactic-centric approach for automating traceability of quality concerns. In *Proceedings of the 2012 Intl. Conf. on Software Engineering (ICSE)*. IEEE, 639–649.

[21] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2013. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 2013 Intl. Conf. on Software Engineering (ICSE)*. IEEE Press, 522–531.