

Automating Software Traceability Link Recovery and Maintenance using Word Embeddings within a Shallow Neural Network

Marlan McInnes-Taylor

Faculty Mentor: Dr. Chris Mills

Florida State University Department of Computer Science

1017 Academic Way, Tallahassee, FL 32304

mcinnest@cs.fsu.edu, crmills@fsu.edu

1 Abstract

In addition to code, software systems contain a multitude of files documenting features, known issues, legal requirements, etc. Software traceability is the ability to link related documents from these various sets through a process called *traceability link recovery*. While previous research has shown that established software traceability improves the quality of projects and makes them easier to maintain, establishing software traceability is often a manual, arduous, and error prone task. This research explores automating the process of software traceability link recovery using established techniques in machine learning. The results demonstrate that even with minimally tuned hyperparameters a shallow neural network can effectively predict which text-based artifacts within a software system are related to one another.

2 Introduction

Previous studies have shown that access to information explaining relationships between artifacts in a system leads to higher quality software and lower bug counts. This is largely credited to such information improving common software engineering tasks such as program comprehension, concept and bug localization, and defect prediction among many others [5, 10, 13]. Unfortunately, the acquisition of traceability information is difficult and typically an afterthought during system construction. Consequently, hundreds or thousands of man hours can be spent after initial development manually inferring relationships between artifacts that are constantly in flux as the system changes during development and maintenance [8, 14, 2]. Therefore, even if a firm is able to establish traceability for a mission critical system, the effectiveness of that information is potentially only temporary. To address this situation, previous studies have attempted to either completely or partially automate the process of establishing traceability links between system components [4]. In this work, we continue that research agenda by using neural networks to model *potential* links between text-based software artifacts and predict which are valid (i.e., two related documents) and which are invalid (i.e., two unrelated documents). Preliminary results of using this technique are promising and show higher recall and precision than its contemporaries [12, 11].

3 Materials and Methods

The training and test data were comprised of six datasets commonly used to validate approaches for automating traceability link recovery: Albergate, Eanci, eTour, iTrust, Modis, and SMOS. Together, these datasets represent XYZ documents and XYZ potential links, XYZ% of which are valid and XYZ% are invalid. Table ABC shows a breakdown of the data by project. Each of the projects involved in our evaluation is written in either Java or C++. The code used for parsing and cleaning the software artifacts is written in Python. TensorFlow 2.0 was used to train and evaluate the neural networks.

3.1 Methodology

Our evaluation consists of a preprocessing, training, and validation phase.

3.1.1 Preprocessing

Document text was first tokenized using NLTK’s[3] punkt tokenizer. Then documents were cleaned by removing stopwords using NLTK’s built in stopwords list augmented with a collection of Java and C++ keywords. Additionally, whitespace, punctuation, and purely numeric tokens were removed from each document. Finally, a Porter Stemmer CITATION was applied to all remaining terms. It is important to note that this stemmer was chosen because it language invariant and our datasets contain both English and Italian words. Each dataset consists of two sets of artifacts of different types (e.g., code classes and use cases), and this same process was applied to documents in both of those sets.

Once the data was cleaned, we created *metadocuments* by concatenating all possible (order invariant) pairs of documents such that the documents in a pair belong to different sets of artifacts. Therefore, each *metadocument* contains data from a unique pair of potentially related documents in the system, each of these pairs is then a potential traceability link by construction. Further, the text of each metadocument was randomly shuffled using the Numpy’s[7] shuffle method, in order to reduce potential bias from token arrangement when setting the sequence length. Finally, each metadocument in our aggregated dataset was labeled as either a valid link between two related documents or an invalid link between two unrelated ones as specified in each dataset’s oracle file. This information was then used to evaluate model predictions.

As shown in Table ABC, class imbalance was present in all datasets due to the large number of metadocuments (i.e., artifact pairs). As one might expect, for a given system there are far fewer metadocuments representing valid links than invalid ones. To mitigate negative effects of class imbalance on model performance, each dataset was balanced using Imbalanced-Learn’s[9] SMOTE implementation. For our evaluation, we continued synthetic metadocument generation until an equal number of valid and invalid links was reached within each dataset.

3.1.2 Training and Validation

In order to construct a minimally complex proof of concept for the approach, a shallow Tensorflow[1] model was implemented classify unlabeled metadocuments as valid or invalid. The model’s first two layers perform text vectorization and transform the vectors using word embeddings. These vectors are then pooled before being passed into a 16 node dense layer followed by the output layer. Based on initial positive results on the SMOS dataset, we kept these hyperparameters and performed 50 trials of 10 fold cross validation with 15 epochs per fold on each dataset. Shuffled stratification via scikit-learn’s[6] StratifiedShuffleSplit implementation was used to sample the dataset in each fold to again minimize selection bias.

4 Results

Dataset	Loss	Accuracy	Recall	Precision	F1Score
Albergate	0.28	0.95	0.91	0.98	0.94
Eanci	0.11	0.96	0.94	0.98	0.96
Etour	0.13	0.96	0.93	0.99	0.96
iTrust	0.17	0.97	0.95	0.99	0.97
Modis	0.29	0.93	0.94	0.92	0.93
SMOS	0.33	0.87	0.75	0.98	0.85

5 Conclusions and Future Work

Table 4 shows the results of our evaluation. Note that here we report recall, precision, and F1 score in addition to accuracy. This is because accuracy can be artificially inflated in imbalanced data. For example, in a dataset of 100 samples with 1 “false” label, a machine that always predicts “true” has high accuracy, but it’s low recall illustrates the machines impracticality. Therefore, given our precision and recall results,

it is clear that word embeddings used within a shallow network can successfully perform metadocument classification for the systems under study. Future work will focus on improving this model by further optimizing its hyperparameters and evaluating the model’s generalizability. Moreover, we plan to investigate minimizing the start-up cost associated with employing our model to make it appealing to industry partners with greenfield systems. Initially, we plan to perform a series of experiments to identify the minimum data requirements using supportive techniques such as Active Learning and cross training.

References

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.
- [2] Giuliano Antoniol, C Casazza, and Aniello Cimitile. Traceability recovery by modeling programmer behavior. In *Proceedings of the Seventh Working Conference on Reverse Engineering*, pages 240–247. IEEE, 2000.
- [3] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O’Reilly Media, 2009.
- [4] Markus Borg, Per Runeson, and Anders Ardö. Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, 2014.
- [5] Elke Bouillon, Patrick Mäder, and Ilka Philippow. A survey on usage scenarios for requirements traceability in practice. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 158–173. Springer, 2013.
- [6] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.
- [7] Charles R. Harris, K. Jarrod Millman, St’efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern’andez del R’io, Mark Wiebe, Pearu Peterson, Pierre G’erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
- [8] Laurence James. Automatic requirements specification update processing from a requirements management tool perspective. In *Proceedings of the International Conference and Workshop on Engineering of Computer-Based Systems*, pages 2–9. IEEE, 1997.
- [9] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.
- [10] Patrick Mäder and Alexander Egyed. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering*, 20(2):413–441, 2015.
- [11] Chris Mills, Javier Escobar-Avila, Aditya Bhattacharya, Grigoriy Kondyukov, Shayok Chakraborty, and Sonia Haiduc. Tracing with less data: Active learning for classification-based traceability link recovery. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 103–113. IEEE, 2019.
- [12] Chris Mills, Javier Escobar-Avila, and Sonia Haiduc. Automatic traceability maintenance via machine learning classification. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 369–380. IEEE, 2018.
- [13] Patrick Rempel and Patrick Mader. Preventing defects: The impact of requirements traceability completeness on software quality. *IEEE Transactions on Software Engineering*, 2016.
- [14] Klaus Weidenhaupt, Klaus Pohl, Matthias Jarke, and Peter Haumer. Scenarios in system development: current practice. *IEEE Software*, 15(2):34–45, 1998.