Inherent Characteristics of Traceability Artifacts

Less is More

Jane Huffman Hayes
Computer Science
University of Kentucky
Lexington, USA
hayes@cs.uky.edu

Giulio Antoniol, Bram Adams, Yann-Gaël Guéhéneuc
GIGL, Polytechnique Montreal
Montreal, Canada
giuliano.antoniol@polymtl.ca, bram.adams@polymtl.ca,
yann-gael.gueheneuc@polymtl.ca

Abstract*—This paper describes ongoing work to characterize the inherent ease or "*traceability*" with which a textual artifact can be traced using an automated technique. Software traceability approaches use varied measures to build models that automatically recover links between pairs of natural language documents. Thus far, most of the approaches use a single-step model, such as logistic regression, to identify new trace links. However, such approaches require a large enough training set of both true and false trace links. Yet, the former are by far in the minority, which reduces the performance of such models. Therefore, this paper formulates the problem of identifying trace links as the problem of finding, for a given logistic regression model, the subsets of links in the training set giving the best accuracy (in terms of G-metric) on a test set. Using hill climbing with random restart for subset selection, we found that, for the ChangeStyle dataset, we can classify links with a precision of up to 40% and a recall of up to 66% using a training set as small as one true candidate link (out of 33) and 41 false links. To get better performance and learn the best possible logistic regression classifier, we must "discard" links in the trace dataset that increase noise to avoid learning with links that are not representative. This preliminary work is promising because it shows that few correct examples may perform better than several poor ones. It also shows which inherent characteristics of the artifacts make them good candidates to learn efficient traceability models automatically, i.e., it reveals their *traceability.

***Index Terms*—Traceability, machine learning, model, logistic regression, artifact characteristics.**

I. Introduction

Traceability is defined as "the ability to follow the life of a requirement, in both a backward and forward direction" [1]. Researchers have explored ways to automatically or semi-automatically recover traceability information [2], [3], [4], [5], [6] from numerous and diverse artifact pairs—source code, requirements, design, bug reports, to name a few. Many advances have been made and a vibrant community has been established, largely anchored by the Center of Excellence for Software and Systems Traceability (COEST) and through the tools and datasets that it provides freely.

Still, traceability information is not routinely generated as a first class citizen of the development lifecycle. Also, when traceability information is generated, it is often not used to its fullest potential and–or is not kept up to date. Further, the tracing task is still largely manual and–or requires that a human analyst performs assurance of generated links or approaches. The overarching goal of reducing the work of the human analyst is not yet fully realized [7].

Recently, a number of researchers have begun applying machine learning techniques and other approaches from software analytics and artificial intelligence to recover traceability information. We concur that such techniques could be used to reduce the work of the analyst, but we wonder if (and to what degree) it is possible to determine the inherent "traceability" of software development artifacts, i.e., how easy (or hard) is it for an automated trace technique to recover links from the artifacts? To explore this question, we revisited an unexplored idea from early in our work on traceability [8].

The rest of the paper is organized as follows. Section II defines the inherent characteristics of traceability information that we study. Section III describes the research approach and progress to date on modeling. Section IV describes our findings to date on *traceability*, Section V presents related work, and Section VI concludes.

II. Inherent Traceability

While traceability has been defined previously as something that is not classified together with standard "–ilities", such as portability, reliability, maintainability, etc., we wonder if there should not also be a definition of traceability as a property or *inherent characteristic* of software artifacts. A traceable artifact would imply that it can easily be traced using automated tools, just as a testable artifact lends itself to easy verification via software testing. For such artifacts, developers and analysts could spend less time on manual traceability, while artifacts that are not as traceable would require much more manual effort to specify or recover trace links.

As the original definition of traceability has not gained widespread use, we wonder if a new name is necessitated for the quality that makes an artifact traceable. We are considering: essential traceability, traceability essence, and traceability. We use the term *traceability*[1] in italics in the remainder of the paper to distinguish it from, yet still relate it to, the typical concept of traceability in the literature. Regardless of the moniker, we want to discover whether inherent characteristics of software artifacts can describe the ease with which artifacts can be traced automatically.

[1] "traceability" is italicized to differentiate it from potentially incorrect uses of the term and to denote our use of it.

Our initial notes from years ago on the *traceability* of artifacts showed that our work stalled due to a lack of data, in two ways. The first way was due to the low volume of available data sets, preventing us from having a sufficiently rich set of case studies that would enable generalization of our findings. This problem essentially has been solved, as a major contribution of the COEST and our international research community is a compilation of many data sets for tracing.

However, the second way in which our earlier work failed still prevails, i.e., the inherent problem that out of the full set of possible trace links between pairs of artifacts (in the worst case quadratic in the number of artifacts), only a minority of the links actually make sense. This creates an imbalance between the number of correct and incorrect links, complicating the task of traceability.

Given that tools and datasets have become readily available, now appears to be the right time for studying the impact of unbalanced data on the *traceability* of software artifacts.

III. Research Approach

To understand whether we can characterize the *traceability* of software artifacts, we ask these research questions:

- Can we learn anything interesting from applying machine learning techniques to various trace datasets?
  - Can we, for example, learn a model that can be applied to a different dataset to predict a link?
- Can we discover the attributes of a dataset (or an artifact pair) that cause it to work well (or not) with automated tracing methods?
- Can we uncover interesting trends such that we can predict trace links accurately if a human user gives us some level of "training" data?
- Can we minimize the training data required?

To address these questions, we undertook four steps. First, we gathered a number of well-used datasets. The current collection of datasets includes: CM-1, MODIS, ChangeStyle, and Etour. Next, we used the RETRO.NET tool [9] and generated candidate trace links (using VSM with tf-idf weighting) for artifact pairs. These candidate links (which consisted of the high-level requirement ID, low-level requirement ID, text of each requirement, and relevance weight of the pair, i.e., the similarity of the given artifacts as computed by the VSM), joined with the answer sets for each dataset, became our input dataset.

Second, we considered ways to characterize the *traceability* of our input dataset. Based on our earlier work in maintenance and software maintainability [10], for this preliminary work, we felt that readability and understandability of text might play an important role. Thus, we studied a number of readability measures that could be obtained from natural-language artifacts. To collect size-based measures such as number of unique words per artifact, we wrote Python scripts. We then considered measures that could be obtained by applying parts of speech (POS) tagging to the artifacts: number of nouns per artifact and number of verbs per artifact.

Finally, we collected readability measures as calculated by a Web app available on-line[2]. The measures collected are shown in Table I. Our hypotheses for the readability / understandability measures were generally: as grade level increases (reading ease decreases), understandability and *traceability* decrease; as reading ease increases, *traceability* increases; as the number of words, sentences or complex words increases, *traceability* increases. We further surmised that if a text element would be too small (few words, few sentences, few syllables), it would be hard to trace. We then took each of the aforementioned measures, whose values were in separate files, and used shell scripts to build a single Weka ARFF file for our input dataset.

In the third step, we applied various WEKA classifiers using 10-fold cross-validation (an evaluation method used when the available data set is small) to assess the recall and precision for predicting True and False links. Precision is the percentage of links suggested by a classifier that are correct (the higher, the less false alarms), while recall is the percentage of all existing correct links that eventually were found by a classifier. Ideally, both values should be high, yet a high value of one usually comes at the expense of a lower value for the other, hence a trade-off must be found.

In general, the number of True links is much lower than the number of False links ("unbalanced data"), which means that when building a classifier for True links, WEKA cannot study enough True links to make a high-performing model, causing the model to identify True links only poorly. On the other hand, a model to find False links would perform much better; in fact just by randomly guessing that a link is False, one would be correct most of the time.

To counter unbalanced data, we tried re-balancing the data by a combination of oversampling (i.e., repeating) True links in our training set and undersampling (i.e., leaving some out) False positive links in our training set. Such re-balancing is common in data mining and aims at having a distribution closer to 50-50 of both kinds of links in the training sets. This resulted in an increase in recall for the True link entries of about 5% but at a cost of dropping precision by about 3%.

At that point, we moved to step four of our approach, which involves statistically investigating the data using R. We carefully studied the ChangeStyle input dataset through an exploratory analysis. ChangeStyle has 31 high-level artifacts (these are requirements) and 17 low-level artifacts (test cases). There are 527 total candidate links, of which 33 are True links. The relevance weight for the True links, computed by RETRO.NET using VSM, is not always high. In fact, there is one True link with a relevance weight of 0.0 and four with one below 0.01.

Our exploration of the ChangeStyle input dataset involved applying feature selection in R to build a logistic regression classifier. We used automatic stepwise variable reduction followed by manual variable pruning to build the most parsimonious logistic regression model to predict true links. The model that was learned on the entire data set of 527 candidate trace

[2] http://read-able.com/

links, for which we knew the correct links, showed that the trace relevance weight was by far the most important characteristic of the artifacts. The next most important characteristics were, for the low-level artifacts, the Flesch-Kincaid Reading Ease (low_fkre), the Flesch-Kincaid Grade Level (low_fkgl), the number of words (low_now), the number of complex words (low_nocw) and, for the high-level artifacts, the Coleman-Liau Index.

TABLE I. READABILITY MEASURES

Abbreviation	Readability Measure	Description/Indication
fkre	Flesch Kincaid Reading Ease	Ease with which a text can be read, higher means easier to read
fkgl	Flesch Kincaid grade level	Grade level of a text, higher means harder to read
gfsc	Gunning Fog Score	Grade level of a text, higher means harder to read
smi	smog index	Estimates years of education needed to read text, higher means harder to read
cli	Coleman Liau Index	Alternative way to measure grade level of a text, higher means harder to read
ari	Automated Readability Index	Alternative way to measure grade level of a text, higher means harder to read
nos	No. of sentences	Complex text elements have many sentences, higher means harder to read
now	No. of words	Complex text elements have many words, higher means harder to read
nocw	No. of complex words	Complex words have many syllables, higher means harder to read
pocw	Percent of complex words	Complex words have many syllables, higher means harder to read
awps	Average word per sentence	Complex sentences have many words, higher means harder to read
aspw	Average syllables per word	Complex words have many syllables, higher means harder to read
gl	grade level	Alternative way to measure grade level of a text, higher means harder to read

Having established that we could obtain a sound logistic regression model, we still faced the problem that there are far fewer True links than False links, something that the rebalancing could only slightly deal with. To solve this problem, we first divided the set of candidate links into a training set (1/3 to 1/5 of candidate links) and a test set (complement of the training). We then formulated the problem of learning a model on the training set as the problem of finding, for a given logistic regression structure, the subsets of links in the training set giving the best accuracy, measured as G-metric on the test set. The G-metric or G-measure is the harmonic mean of recall and specificity (i.e., the percentage of False links found).

We then implemented the link subset selection optimization problem as a hill climbing approach with random restart. At each move, the hill climbing attempts to improve the accuracy (measured as G-metric) by removing a (randomly selected) training datum, moving it into the test set, and (1) learning a new logistic classifier and (2) evaluating the logistic classifier on the test set. The logistic classifier is implemented with fixed structure described above and a simple threshold of 0.5. Probabilities above 0.5 are considered True links. At each step, the search is guided only by the G-metric, although the search for the best possible combination is performed on a pruned set of candidate links: we discard all candidate links with a relevance weight lower than 0.001, as these links seldom correspond to correct links. When after a certain number of moves the solution is not improved, the Hill climbing with random restarts saves the current solution and starts a new search from a new initial random solution in order to avoid local optima.

We found that for the ChangeStyle input dataset, we could classify links with a precision of up to 83% and a recall of up to 42% using a training set as small as nine True candidate links (out of 33 correct links, 27%) and 136 False links (out of 503 false links). However, if we accepted a recall of 50% and a precision of 40%, two True links and 27 False links suffice.

Thus, we conclude that, to get better performance and learn the best possible logistic regression model, we must "discard" artifacts in the trace dataset that increase noise (links with low relevance weight) to avoid learning on links that are not really representative of the actual true links in the data set. Next, we discuss the ties between these results and traceability.

IV. INHERENT CHARACTERISTICS OF ARTIFACTS

This section examines the measures of interest in our model to answer our questions and understand *traceability*.

To understand the artifacts' *traceability*, we run several experiments using the hill climbing search. We configured the hill climbing with random restart in the following way. In each experiment, the algorithm performs 50 restarts. Each search starts from an initial solution made up by, at most, one third of the trace links; the actual number is randomly chosen between one and one third of the True and False links. At each step, for a given solution, the algorithm attempts 20 times to improve the solution moving one datum, randomly chosen, from training to test; if no improvement is found, the search is restarted from a newly generated random solution. Thus each experiment produces 50 pairs of training and test data sets, sets of

various sizes and with different trace link classification accuracy. We run 20 experiments and then collect and inspect the data for an overall of 1000 set pairs.

Figure 1 (shown at the end of the paper to permit a full page depiction) shows paired scatterplots of the variables included in the obtained logistic models (relevance_weight, low_fkre, low_fkgl, low_now, low_nocw, and high_cli). Each box plots one variable against the other. The upper right box depicts relevance_weight against high_cli. The grey dots are the False links, the red dots and green dots are True links. The figure shows a special subset of found solutions: solutions containing only one or two True links where the G-measure is over 60. These are quite puzzling solutions as just a handful of True candidate links (one or two green dots) performs better (sometimes much better) than solutions containing many more True links. In Fig. 1, true positive links never selected as part of the training set are colored in red, green dots are those True links that were retained as part of the training set. Surprisingly, for these special cases, the selected true candidate links are links that do not have the highest possible relevance weight. However, they tend to have a high value of low_fkre: they should be easy to understand. The algorithm seems to prefer solutions with average values of high_cli and low_now (and low_nocw), see lower right quadrant of Fig. 1.

It appears at first glance that relevance_weight, low_now, and high_cli separate the two types of data (False and True links). Further, it seems that a preliminary conclusion from the plots for low_fkre is that for artifacts to be traceable, their fkre values should be above 48.45, i.e., "easily understood by 15- to 21-year-old students."[3] This is confirmed by data reported in Table II. Table II reports the values for three "green" data points of Fig. 1. It appears that the best way to predict a trace link is to select true candidate links that do not have a high relevance weight but a low_fkre of 48.9 or higher and avoid extreme values of low_now or low_nocw.

Translating this observation into guidelines for artifacts' authors, increased fkre, i.e., reading ease, increases *traceability*. Oddly, high relevance weight does not necessarily imply *traceability*. A small or very high number of words in the low-level artifacts decrease *traceability*, as expected. The same guideline is true for complex words in the low-level artifacts. A moderately high reading level leads to good *traceability*, but if it is too high/low, *traceability* decreases. This makes sense because a text written at, say, a 3rd grade-reading level will likely use simple, common words and may trace to everything while a text written at a post-graduate level may be too complex and thus hard to trace.

There are many threats to the validity of our work; most notable at present are the external threats, as we only report on one small dataset in one domain and for one project. Our work is preliminary and we plan to evaluate other datasets in varied domains. In addition, our ongoing work will include stringent empirical validation of our ideas.

[3] http://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests#Flesch_Reading_Ease

V. Related Work

There was no prior work on *traceability* of artifacts, yet maintainability and testability provide excellent analogies. Maintainability is the "ease with which maintenance can be performed" [11]. Welker defined a model for measuring maintainability of source code called the maintainability index (MI) [12]. This model, built using regression analysis, combines various static measures, such as McCabe's cyclomatic complexity [13] and Halstead's software science measures [14] to indicate how easy it will be to maintain code. Yu et al. [11] discuss sources of data for measuring maintainability of open-source projects, specifically data from defect tracking systems, change logs, and source code. They also point out challenges in trying to use such data to measure maintainability. Voas described a dynamic method for estimating the testability of source code (ease with which faults can be detected) using propagation, infection, and execution [15]. Khan and Mustafa identified a set of metrics for object-oriented code that can predict the testability of classes; they found that their model could identify problem areas, thus improving the quality of the product and decreasing the required testing effort [16].

In related traceability work, Guo et al. proposed the notion of a domain-specific traceability solution where they model the high-level reasoning undertaken by a human analyst [17]. The work features a rule set for extracting action units from the artifacts being traced, a knowledge base of semantic concepts for the domain, and link creation heuristics. The work is similar to ours in that it models the artifacts to be traced and seeks to make avail of semantic concepts. It differs from our work in that we model characteristics of the artifacts that we believe to contribute to ease of tracing with an automated tool.

VI. Conclusions

This is initial and ongoing work, but it shows promise: few correct examples may perform better than several poor examples. This work is a proof of concept that a few manually labeled true trace links (in the extreme case one or two) and a moderate number of false trace links (few dozens) are enough to obtain acceptable preliminary results (on about 500 links). Our current answers to our research questions, based on just one dataset, are:

- *Can we learn anything interesting from applying machine learning techniques to various traceability datasets? Can we learn a model that can be applied to a different dataset to predict a link?* The early finding on this is yes, we can learn interesting things and we can possibly predict links. We have yet to apply our model to different datasets.
- *Can we discover the attributes of a dataset (or an artifact pair) that cause it to work well (or not) with automated tracing methods?* Yes, early results indicate (for one dataset) that relevance_weight, low_fkre, low_fkgl, low_now, low_nocw, and high_cli are attributes that impact the *traceability* of an artifact.
- *Can we uncover interesting trends such that we can predict trace links accurately if a human user gives us some level of "training" data?* Yes, our latest results indicate that we are able to achieve decent results (recall of 42%

TABLE II. THREE DATAPOINTS WITH G-MEASURE OVER 60

relevance_ weight	high_tot_ words	high_uniq_ words	link	low_tot_ words	low_uniq_ words	low_ gl	low_ fkre	low_ fkgl	low_g fsc	low_s mi	low_cli
0.091701	13	11	yes	176	116	14	48.9	13.8	14.8	10.6	12.6
0.002842	7	7	yes	137	91	8	67	7.1	7.7	6.5	11.3
0.091037	12	12	yes	176	116	14	48.9	13.8	14.8	10.6	12.6

low_ari	low_nos	low_now	low_nocw	low_pocw	low_awps	high_ gl	high_ fkre	high_fkgl	high_gfsc	high_ smi
16.2	6	179	20	11.17	29.83	9	63.5	7.6	5.2	8.3
6.6	11	140	13	9.29	12.73	6	54.7	6.6	1.6	4.4
16.2	6	179	20	11.17	29.83	10	46.6	9.7	8.1	10.1

high_cli	high_ari	high_nos	high_ now	high_ nocw	high_pocw	high_ awps	high_noun_ count	high_verb_ count	low_noun_count	low_verb_count	
15.9	10.4	1	13	2	15.38	13	6	0	71	8	
12.8	3.5	2	8	1	12.5	4	3	2	71	7	
14.1	8.5	1	12	3	25	12	4	0	71	8	

and precision of 83%) with nine links (27% of the true links) and 136 false links provided by the human user, for one dataset.

- *Can we minimize the training data required?* Possibly, we found that with just two True links (6%) and 27 False links, a recall of 50% and a precision of 40% can be achieved.

Future work will be focused on identifying those sets of features leading to selection of the best possible representative elements in a link dataset to build the most accurate and parsimonious model without the need to evaluate the model on the test set.

ACKNOWLEDGMENT

We thank NASA and NSF for prior partial funding of ideas presented here under grants NAG5-1173 and CCF-0811140.

REFERENCES

[1] O. Gotel and A. Finkelstein, "Extended Requirements Traceability: Results of an Industrial Case Study," in *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, 1997, p. 169.

[2] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering Traceability Links between Code and Documentation," *IEEE Trans. Softw. Eng.*, vol. 28, no. 10, pp. 970–983, 2002.

[3] A. Marcus and J. Maletic, "Recovering Documentation-to-Source Code Traceability Links using Latent Semantic Indexing," in *Proceedings of the Twenty-Fifth International Conference on Software Engineering, ICSE 2003*, 2003, pp. 125–135.

[4] J. Cleland-Huang, C. K. Chang, G. Sethi, K. Javvaji, H. Hu, and J. Xia, "Automating speculative queries through event-based requirements traceability," in *Proceedings of the IEEE Joint International Requirements Engineering Conference (RE '02)*, 2002, pp. 289–296.

[5] J. H. Hayes, A. Dekhtyar, and J. Osborne, "Improving Requirements Tracing via Information Retrieval," in *International Conference on Requirements Engineering, Monterey, California*, 2003, pp. 151–161.

[6] Yadla, Suresh, Hayes, Jane Huffman, and Dekhtyar, Alex, "Tracing requirements to defect reports: an application of information retrieval techniques," *ISSE*, vol. 1, no. 2, pp. 116–124, 2005.

[7] J. Cleland-Huang, O. Gotel, and A. Zisman, Eds., *Software and Systems Traceability*. London: Springer London, 2012.

[8] J. H. Hayes, A. Dekhtyar, S. Sundaram, and S. Howard, "Helping Analysts Trace Requirements: An Objective Look," in *International Conference on Requirements Engineering (RE'2004)*, 2004.

[9] J. Hayes, A. Dekhtyar, S. Sundaram, E. Holbrook, S. Vadlamudi, and A. April, "REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery," *Innov. Syst. Softw. Eng.*, vol. 3, no. 3, pp. 193–202, Sep. 2007.

[10] J. H. Hayes and J. Offutt, "Recognizing Authors: An Examination of the Consistent Programmer Hypothesis," *Softw Test Verif Reliab*, vol. 20, no. 4, pp. 329–356, Dec. 2010.

[11] L. Yu, S. R. Schach, and K. Chen, "Measuring the maintainability of open-source software," in *2005 International Symposium on Empirical Software Engineering, 2005*, 2005, p. 7 pp.–.

[12] K. D. Welker, P. W. Oman, and G. G. Atkinson, "Development and Application of an Automated Source Code Maintainability Index," *J. Softw. Maint.*, vol. 9, no. 3, pp. 127–159, May 1997.

[13] T. J. McCabe, "A Complexity Measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.

[14] M. H. Halstead, *Elements of Software Science*. Elsevier, 1977.

[15] J. M. Voas, "PIE: A Dynamic Failure-Based Technique," *IEEE Trans Softw Eng*, vol. 18, no. 8, pp. 717–727, Aug. 1992.

[16] R. A. Khan and K. Mustafa, "Metric Based Testability Model for Object Oriented Design (MTMOOD)," *SIGSOFT Softw Eng Notes*, vol. 34, no. 2, pp. 1–6, Feb. 2009.

[17] J. Guo, N. Monaikul, C. Plepel, and J. Cleland-Huang, "Towards an Intelligent Domain-specific Traceability Solution," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, New York, NY, USA, 2014, pp. 755–766.

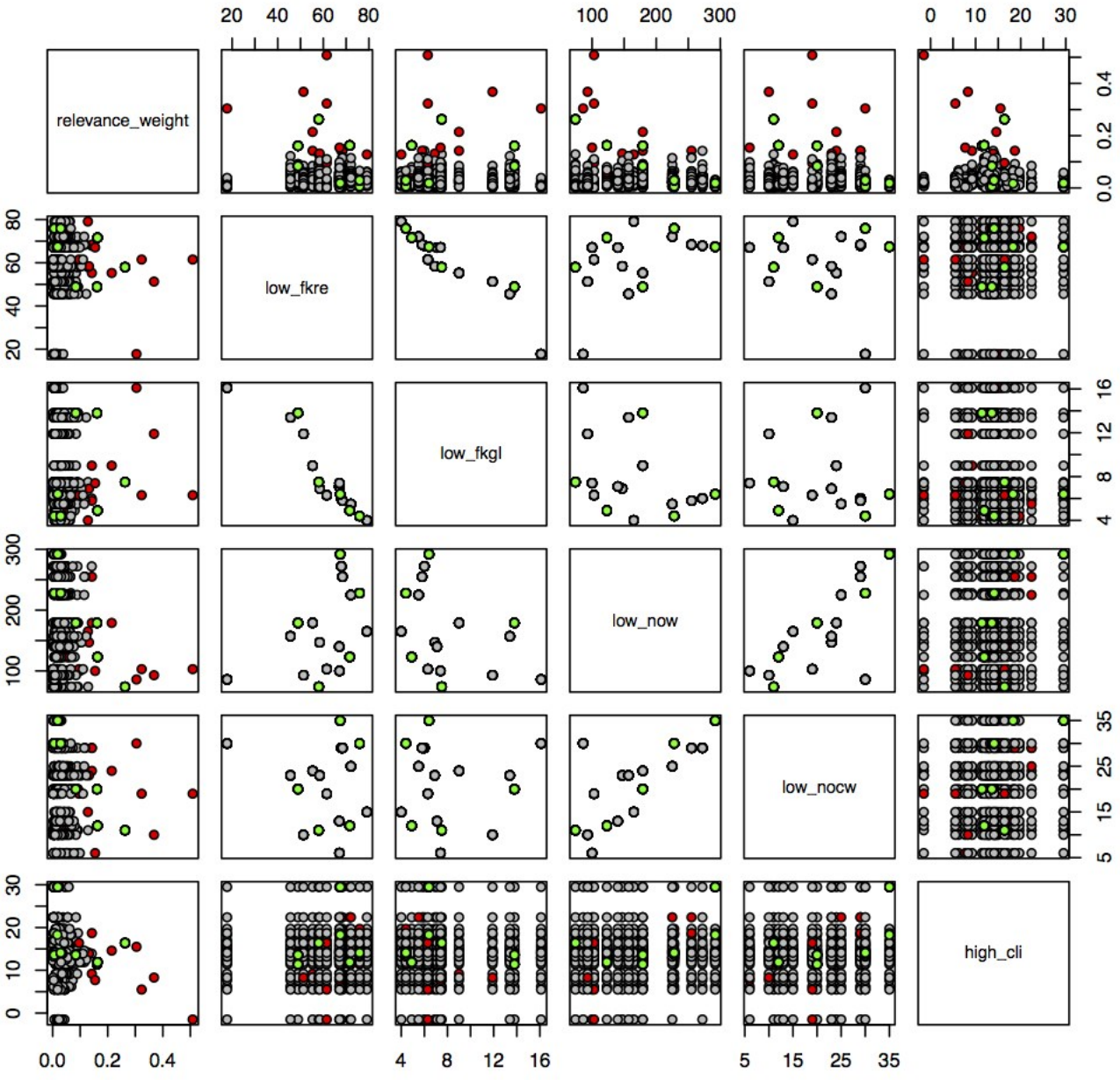

Fig. 1. Scatterplot for ChangeStyle dataset