Weka Meets TraceLab: Toward Convenient Classification

Machine Learning for Requirements Engineering Problems: A Position Paper

Jane Huffman Hayes, Wenbin Li
Computer Science Department
University of Kentucky
Lexington, Kentucky, USA
{hayes,li}@cs.uky.edu

Mona Rahimi
School of Computing
DePaul University
Chicago, Illinois, USA
m.rahimi@acm.org

***Abstract*—Requirements engineering encompasses many difficult, overarching problems inherent to its subareas of process, elicitation, specification, analysis, and validation. Requirements engineering researchers seek innovative, effective means of addressing these problems. One powerful tool that can be added to the researcher toolkit is that of machine learning. Some researchers have been experimenting with their own implementations of machine learning algorithms or with those available as part of the Weka machine learning software suite. There are some shortcomings to using "one off" solutions. It is the position of the authors that many problems exist in requirements engineering that can be supported by Weka's machine learning algorithms, specifically by classification trees. Further, the authors posit that adoption will be boosted if machine learning is easy to use and is integrated into requirements research tools, such as TraceLab. Toward that end, an initial concept validation of a component in TraceLab is presented that applies the Weka classification trees. The component is demonstrated on two different requirements engineering problems. Finally, insights gained on using the TraceLab Weka component on these two problems are offered.**

***Index Terms*—Artificial intelligence, machine learning, requirements engineering, classification, decision trees, TraceLab, Weka.**

I. Introduction

Challenges loom large as we endeavor to develop software, many of these related to requirements engineering (RE) and its subareas of process, elicitation, specification, analysis, and validation [1]. We encounter obstacles as we elicit requirements, attempting to ensure that all stakeholders are represented. We encounter a gap as we move from the problem space (requirements) to the solution space (design); we struggle to specify requirements that are complete and consistent. As researchers, we strive to understand these and other issues and to develop beneficial approaches.

At the heart of many of these requirements challenges lies a sub problem of pattern recognition - the problem of classification: finding "a mapping from unlabeled instances to (discrete) classes. [2]" Ensuring that all stakeholder groups have been represented in elicitation can be accomplished by classifying each elicited requirement by its stakeholder type and ensuring coverage. Verifying requirement allocation to design, as part of bridging the gap, may require that requirements be classified as satisfied or not satisfied by their allocated design elements. Analyzing requirements to ensure completeness and consistency may necessitate putting requirements into various categories typically found in requirement specifications to support a high level check that at least one requirement exists in each category. For instance, a medical device must have a set of safety and performance requirement specifications to guarantee it is safe to use. Similarly, health care management systems require the specifying of requirements which guarantee privacy and availability of patient data. Requirement elicitation for an air traffic control system must include requirements expressing the necessity of the fast response time of the system. These important activities have a common denominator: classification.

In this paper, we posit that a class of problems exists in requirements engineering that lend themselves well to machine learning techniques, specifically to the collection of supervised learning methods termed classification trees, in Weka [3]. Examples include separating functional and quality-focused concerns to facilitate further assessment of the system or extracting only performance related quality concerns to specify the response time constraints in the system. Further, we posit that a barrier to adoption of such techniques may be the lack of easy to use tools that are aimed at requirements engineering researchers and are integrated into research tools (see Section II). We present a component for the TraceLab research framework [4] that makes the Weka classification tree algorithms accessible (Section III). We present two applications of the component as initial concept validation: classification of requirements into ten different security categories and classification of requirements as temporal/non-temporal (Section IV). Finally, we share insights gained through this investigation and conclude (Section V).

II. Background and Position

Artificial intelligence (AI) encompasses a number of areas of study, including cognitive science, machine learning,

representation and reasoning, to name a few. Machine learning is the focus of this paper, defined as "field of scientific study that concentrates on induction algorithms and on other algorithms that can be said to ``learn. [2]" Machine learning algorithms can be further broken down into supervised, unsupervised, semi-supervised, and reinforcement learning. Supervised learning is of interest here.

In supervised learning, a training set, which has labelled instances, is used to determine the class or category of unlabeled instances. The training set contains information on the features or attributes of the instances. These features can be quantifiably represented, as integer, real, ordinal, categorical data, for example. Of the many supervised learning methods, decision trees (classification trees in Weka) combine interpretability, efficiency, and accuracy [5] and are of interest.

A decision tree can be seen as a collection of binary tests organized in a tree structure. The non-terminal nodes of a tree are labeled with tests, comparing the value of an input feature or attribute to a threshold. The terminal tree nodes are labeled with a class. A classification for a new instance (whose attribute values are known) is determined by propagating it into the tree from the top node per the test answers [5]. "When a terminal node is reached, its corresponding class label is attributed to the instance. [5]"

A number of tools and techniques have been introduced to assist with machine learning and classification: Orange, R, JBoost, RandomForests [5]. Perhaps the best known is Weka.

Weka [3] is open source software which contains a collection of data mining procedures including preprocessing, classification, clustering, feature selection, and visualization. Weka is widely used by researchers because it is free, it runs on most platforms, and it supports several data mining tasks. Weka's main user interface is the Explorer, but the same functionality can be also accessed through the component-based Knowledge Flow interface and from the command line.

Weka supports classification by providing a number of algorithms, mainly decision trees. The most well-known algorithms include: J48, which is the java implementation of the popular decision tree algorithm C4.5; Random Forest, which is an ensemble learning method performing the classification task by constructing a multitude of trees and considers the mode of all trees output as the final label; SimpleCART(Classification And Regression Trees), which is a combination of classification trees where the outcome of the tree is the class label and regression trees, where the outcome is a numerical value.

As mentioned in Section I, there are a number of challenges in the subareas of requirements engineering that lend themselves well to classification techniques. For example, often it is necessary to classify or categorize requirements or sub-requirements prior to undertaking an analysis or validation activity. There has been prior success in applying artificial intelligence techniques to requirements engineering problems. Pohl et al. used the Novel Approaches to Theories Underlying Requirements Engineering (NATURE) framework to address five problems areas: process guidance, process traceability, system knowledge acquisition, specification reuse, and requirement definition and critique [7]. Huang et al. successfully used an iterative approach for training and retraining a classifier for non-functional requirements [8]. Sultanov and Hayes used reinforcement learning to perform requirements tracing [9]. Niu et al. used foraging theory, as a form of collective intelligence, to address the problem of assisted requirements tracing [10].

The above researchers, and others, have begun to apply AI techniques to these problems using tools such as Weka or "one off" tools developed specifically for the purpose. There are several disadvantages to this. First, stand-alone solutions do not lend themselves well to replication or reuse. Dit et al. enumerate the advantages of using a research tool such as TraceLab to assist with reuse and replication [6]. Second, TraceLab provides complex constructs for organizing components, including decision and looping constructs, Weka does not. Third, TraceLab permits the development of composite components. Finally, Weka does not have a "market" or community where developers can contribute their components, experiments, or full packaged experiments (with components, types, datasets, results); TraceLab does [6]. These realizations lead to our position.

Position: That machine learning algorithms, specifically supervised learning decision trees, can be used to address problems in requirements engineering.

Further: That wide spread adoption of machine learning algorithms necessitates the integration of the algorithms into research tools for requirements engineering researchers.

Toward this end, we introduce a TraceLab component for applying Weka classification trees; it is discussed next.

III. WEKA MEETS TRACELAB

TraceLab is a tool that assists researchers in designing and executing traceability experiments. It was developed primarily at DePaul University and the College of William and Mary in conjunction with other Universities such as Kent State University and the University of Kentucky. TraceLab is free for download at www.coest.org. The researchers who developed TraceLab also focus on collecting and organizing datasets. Nine datasets related to software requirements are available on the website.

In TraceLab, an experiment is represented as a precedence graph of components, with support provided for basic control flow (as mentioned above). Each component implements a task, such as importing data, pre-processing, tracing documents, and measuring the quality of answer sets. Components exchange data through their inputs and outputs. In a typical traceability experiment, data is imported from external sources (csv files, xml files, etc.) through an "Importer" component and is stored in the TraceLab "workspace." The data may be stored in pre-defined data types representing artifacts. When a component inputs the data, it is loaded from the workspace to that component. Similarly, outputting the data means that the component stores the data back to the workspace.

TraceLab is highly reusable and expandable. Researchers can easily design or reuse existing experiments. The TraceLab

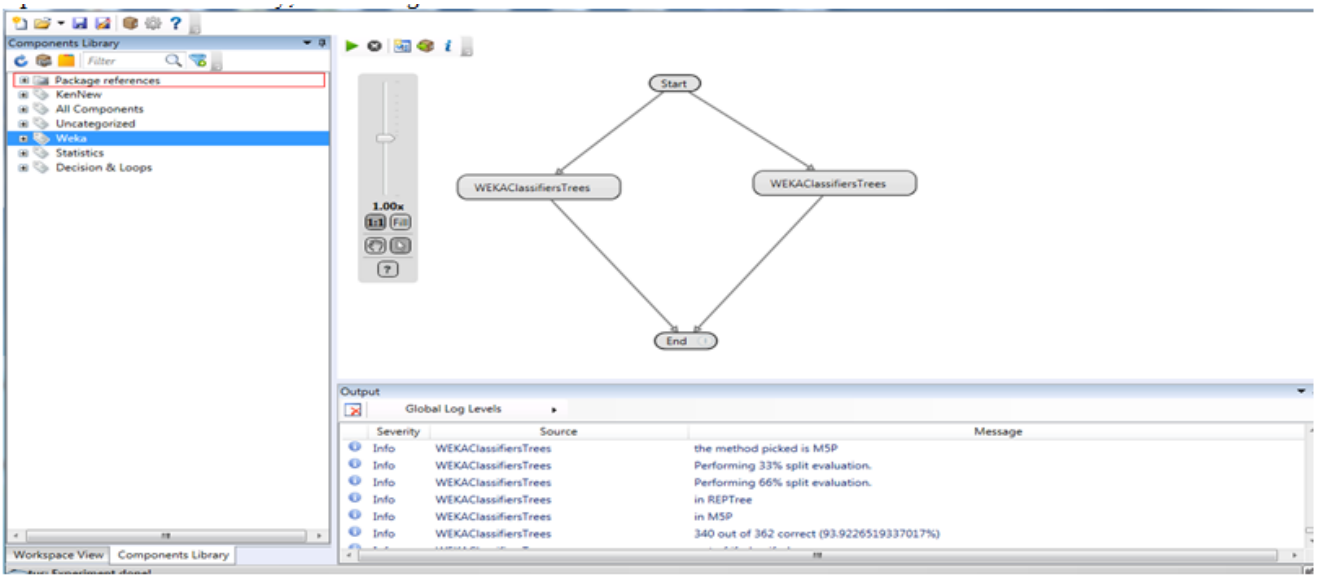

Fig. 1. TraceLab: Comparing classifiers for one dataset.

component library supports common tasks in traceability, feature location, and even software testing experiments. TraceLab is being used for far more than traceability research.

Figure 1 shows the user interface of TraceLab. The component library is on the left side. The Experiment window and the Log window are on the right side. If researchers cannot find the materials they want, they can create their own components or even data types. The new materials can easily be added to the library and shared with other researchers. Researchers can construct a new experiment by selecting components from the library, connecting them based on the order in which components are executed, and setting up the inputs and outputs for each component. If researchers want to reuse an experiment with slight modifications, they only need to replace or reconfigure components of interest. For instance, given an experiment that measures the performance of a tracing algorithm, researchers can replace the tracing component with the algorithms they want to use.

Researchers can also standardize the evaluation by creating benchmarks in TraceLab. A benchmark focuses on a specific task, uses fixed datasets, and has measures to evaluation the efficiency of the task. Researchers can use benchmarks to evaluate the efficiency of multiple algorithms for the same task.

It is intuitive to design experiments with the TraceLab visual environment. TraceLab also allows researchers to execute experiments via the command line: more efficient if researchers need to run a large number of experiments.

The component that we developed is called WekaClassifiersTrees. It is written in C# and consists of roughly 200 lines of code. The component uses the Weka dll. The component can be configured: to specify the .arff file for processing, to select the classification tree for the given dataset, and to specify the test/data split. The component outputs the progress of the processing as well as the results in the Log window of TraceLab. The results that are output are the number of elements correctly classified out of the total number of elements as well as the correctness percentage. Researchers can perform machine learning tasks with this component even if they do not know Weka. The component hides all the details from them. All they need to do is create an experiment as is shown in Figure 1.

IV. RE Applications

In addressing challenges in requirements engineering, the need to classify or categorize requirements or other artifacts may arise. As mentioned in Sections I and II, classification has been shown to be an integral part of elicitation and prioritization, analysis, tracing, and validation.

As initial validation of our position, we examine two requirements engineering problems. First, we classify requirements into functional or non-functional categories: F=Functional, AC=Access Control, PA=Person Authentication, SED= Security Encryption Decryption, AUD=Audit Control, AL=Automatic Logoff, IC=Integrity Controls, UUI=Unique User Identification, TED= Transmission Encryption Decryption, EAP=Emergency Access Procedure and TS= Transmission Security. We demonstrate the TraceLab component by using the Certification Commission for Healthcare Information Technology (CCHIT) dataset [11]. CCHIT is a system for managing electronic healthcare records and consists of 1064 requirements.

In order to undertake this study, we built an arff file (WEKA file format) from the CCHIT dataset, then a set of preprocessing steps are applied to the requirement specifications to convert them into the proper format for classification. First, all common words (stop words) such as "and" or "the" are removed from the requirement specifications and all remaining words are stemmed to their root form. Later, each requirement specification will be converted to a vector of words, $v_i = \{f_{i,1}, f_{i,2},..., f_{i,w}\}$ where $f_{i,j}$ is a term weight representing the number of word j occurrence in requirement i. Finally, we designed a very simple experiment in TraceLab

including our developed component, WEKAClassifiersTrees. We applied two different classifiers, REPTree and J48 (we applied all the available classification trees, but show just these two) by selecting the classification trees and the ratio of interest to decompose the dataset into training and testing sets. As can be seen in Figure 1, we were able to accurately classify 93.92% of the lower accuracy). Furthermore, using multiple instances of the developed component facilitated the comparison of results of applying different classifiers to the same dataset. A subset of the CCHIT dataset is shown in Figure 2.

Second, we classify requirements as temporal or non-temporal to support consistency checking. The 511 dataset has been used for this demonstration. This dataset consists of the system requirements for the Bay Area 511 Regional Real-Time Transit Information System (available open source) [12]. We show a small subset of the 511dataset as an arff file in Figure 3.

```
4,'The system shall associate (store and
link) key identifier information (e.g.
system ID medical record number) with each
patient record.',IC
5,'The system shall be able to support the
standards identified and recommended by the
Health Information Technology Standards
Panel (HITSP) on its HITSP-TP13 Ver 1.0.1
document',F
```

Fig. 2. CCHIT dataset arff sample.

We applied the same process described for the CCHIT dataset to the 511 dataset. We correctly classified 93.12 % of the temporal requirements using the REPTree classification tree (J48 had lower accuracy).

```
1,'Transit agency system generates
predictions periodically',T
2,'The 511 system updates prediction data
that is received from each of the transit
agency systems within 20 seconds after
receiving the prediction data.',T
```

Fig 3. 511 dataset arff sample.

V. WHAT NEXT?

Based on the two presented RE problems, we found that the WekaClassifiersTrees TraceLab component made classification easy, efficient, and repeatable. We used the same component for both problems. We can imagine many other RE problems that might benefit from this component: determining who wrote a specific requirement when a requirement is being changed or is erroneous; classifying requirements as error prone; classifying requirements as change prone; classifying requirements as ambiguous; etc.

Therefore, we feel that we have provided initial validation of our position: that machine learning algorithms can assist with RE problems, and that it is important to integrate these methods into research tools such as TraceLab.

For future work, we plan to develop additional components to facilitate the use of TraceLab for a broad range of RE problems. These helper components will include: arff file builder, tree visualizer, convertor (such as string to wordvector), etc. We plan to make our arff datasets part of the benchmarks for TraceLab available through the Center of Excellence for Software and System Traceability at www.coest.org.

ACKNOWLEDGMENT

The TraceLab development work was funded by the National Science Foundation under grant ARRA-MRI-R2 500733SG067. We thank the Weka developers. We thank Jane Huang for assistance with datasets.

REFERENCES

[1] "Guide to the software engineering body of knowledge: 2004 version," Library of Congress Online Catalog, 2005, http://lccn.loc.gov/2005921729.

[2] Thomas G. Dietterich (Ed.). 1998. Special issue on applications of machine learning and the knowledge discovery process. Mach. Learn. 30, 2-3 (February 1998).

[3] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian Witten (2009) The Weka Data Mining Software: An Update; SIGKDD Explorations, Vol 11, Issue 1.

[4] Jane Cleland-Huang, Adam Czauderna, Alex Dekhtyar, Olly Gotel, Jane Huffman Hayes, Ed Keenan, Greg Leach, Jonathan Maletic, Denys Poshyvanyk, Youghee Shin, Andrea Zisman, Giuliano Antoniol, Brian Berenbach, Alexander Egyed, and Patrick Maeder. 2011. Grand challenges, benchmarks, and TraceLab: developing infrastructure for the software traceability research community. In Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '11). ACM, New York, NY, USA, 17-23.

[5] Pierre Geurts, Alexandre Irrthum, Louis Wehenkel, Supervised learning with decision tree-based methods in computational and systems biology. Molecular BioSystems. 12/09, 5(12):1593-605.

[6] Bogdan Dit, Evan Moritz, Mario Linares-Vasquez, Denys Poshyvanyk, "Supporting and Accelerating Reproducible Research in Software Maintenance Using TraceLab Component Library," 2013 IEEE International Conference on Software Maintenance, pp. 330-339, 2013 IEEE International Conference on Software Maintenance (ICSM), 2013.

[7] Klaus Pohl , Petia Assenova , Ralf Doemges, Paul Johannesson, Neil Maiden, Véronique Plihon, Jean-Roch Schmitt, Giwrgos Spanoudakis, 'Applying AI Techniques to Requirements Engineering: The NATURE Prototype', NATURE Report 94-7.

[8] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. 2007. Automated classification of non-functional requirements. Requir. Eng. 12, 2 (May 2007), 103-120.

[9] Hakim Sultanov, Jane Huffman Hayes: Application of reinforcement learning to requirements engineering: requirements tracing. RE 2013: 52-61.

[10] Nan Niu, Anas Mahmoud, Zhangji Chen, and Gary Bradshaw. 2013. Departures from optimality: understanding human analyst's information foraging in assisted requirements tracing. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13). IEEE Press, Piscataway, NJ, USA, 572-581.

[11] CCHIT dataset, www.coest.org

[12] Real-Time Transit Information System Requirements, http://www.mtc.ca.gov/planning/tcip/Real-Time_TransitSystemRequirements_v3.0.pdf