Tracing with Less Data: Active Learning for Classification-Based Traceability Link Recovery

Chris Mills, Javier Escobar-Avila, Aditya Bhattacharya, Grigoriy Kondyukov, Shayok Chakraborty, Sonia Haiduc
Department of Computer Science
Florida State University
Tallahassee, USA
{cmills, escobara, abhattac, kondyuko, shayok, shaiduc}@cs.fsu.edu

***Abstract*—Previous work has established both the importance and difficulty of establishing and maintaining adequate software traceability. While it has been shown to support essential maintenance and evolution tasks, recovering traceability links between related software artifacts is a time consuming and error prone task. As such, substantial research has been done to reduce this barrier to adoption by at least partially automating traceability link recovery. In particular, recent work has shown that supervised machine learning can be effectively used for automating traceability link recovery, as long as there is sufficient data (i.e., labeled traceability links) to train a classification model. Unfortunately, the amount of data required by these techniques is a serious limitation, given that most software systems rarely have traceability information to begin with. In this paper we address this limitation of previous work and propose an approach based on active learning, which substantially reduces the amount of training data needed by supervised classification approaches for traceability link recovery while maintaining similar performance.**

***Index Terms*—software traceability, machine learning, active learning, classification**

I. Introduction

Software systems are made up of many different sources of information such as source code, bug reports, requirements, use cases, test cases, etc. These different types of software artifacts contain important information about various aspects of the system. *Software traceability* is a system property that represents the degree to which the relationships between related software artifacts of different types are known and documented. For example, in systems with a high level of software traceability, it is known which code segments implement which use cases, which bugs are related to which features, which features cover which requirements, and so forth. Previous work has shown that the information traceability provides natively supports various software tasks such as program comprehension, bug localization, impact analysis, ensuring test coverage, etc. and leads to more reliable projects with better code and fewer bugs [1]–[3]. Unfortunately, collecting this information is often of secondary concern to the actual development, maintenance, and evolution of the project. Therefore, traceability is often established and updated post-hoc, long after artifacts are created or modified. The process of establishing links in this scenario is called *traceability link recovery (TLR)*, and when performed manually it is extremely costly. Further, even if significant resources are invested to establish traceability, it will rapidly degrade as the software system changes due to evolution and maintenance tasks [4]–[7]. Therefore, equally important as establishing traceability is the process of maintaining it as the system changes over time.

Many works have proposed methods for at least partially automating TLR in order to mitigate its high cost, consequently reducing the primary barrier to adoption many projects face. Recent work has focused on automating TLR using supervised machine learning, re-envisioning the task as a binary classification problem [8], [9]. Using this approach, all possible links between two sets of artifacts are first represented in a feature space, and then they are classified by a predictive model that labels each link as *valid* (i.e., the two artifacts in the link are in fact related) or *invalid* (i.e., there is no relevant relationship between the two artifacts in the link). While achieving better results and more automation than traditional approaches to TLR, the classification-based approach also has a glaring limitation: the predictive models require large amounts of existing traceability data for training (up to 90% of all potential links in a system).

Given the fact that most software projects outside of high-risk, safety critical domains do not capture software traceability information, the large amounts of training data required by state-of-the-art classification approaches for TLR are not available in the majority of systems. Therefore, to make a classification-based approach for TLR widely applicable, we must strive to create accurate predictive models that require as little training data as possible. To that end, we introduce ALCATRAL (Active Learning for ClAssification-based TRAceability Links), the first approach to combine active learning with classification-based TLR. *Active learning* is an approach to iteratively train machine learning models by identifying unlabeled data that, once labeled, will teach a model the most about the distinction between classes. In the case of traceability link recovery, this translates into intelligently determining which links a developer needs to label next such that the model learns the most about the difference between valid and invalid links. This process allows for training a model with considerably less training data and prevents the developer from wasting time inspecting and labeling traceability links that do not contribute to the model's predictive performance.

We compared ALCATRAL with our state-of-the-art approach in classification-based TLR called TRAIL [9]. The

2576-3148/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSME.2019.00020

results of the evaluation on 11 datasets show that ALCATRAL provides a predictive model that is not only accurate in predicting the validity of traceability links, but also requires much less training data than TRAIL.

This work makes the following contributions to the state-of-the-art in machine learning classification approaches to TLR:

1) A novel approach, called ALCATRAL, which is the first use of active learning for classification-based TLR
2) A replication of the TRAIL framework trained with datasets of various sizes.
3) An evaluation comparing ALCATRAL with TRAIL, which showed that ALCATRAL is able to achieve performance similar to TRAIL while reducing the amount of training data needed by more than 50%.

The rest of this paper is organized as follows. Section II provides background information on active learning and an overview of related work in TLR. Section III introduces our approach ALCATRAL. Section IV outlines the research questions in our evaluation and the evaluation design for addressing each one. SectionV presents the results of the empirical evaluation. Section VI discusses the threats to the validity of our conclusions and lastly, section VIII provides concluding remarks and a discussion of future work.

II. Background and Related Work

Software traceability in general has been extensively studied by the software engineering research community. While Information Retrieval (IR) approaches account for the bulk of the work toward the automation of TLR [10], [11], in this section we focus on discussing the related work using machine learning approaches for TLR, as they are the most related to our approach. In addition, we also present some background information on *active learning* and its use in the field of machine learning.

A. Machine Learning for Traceability Link Recovery

The earliest work using machine learning for TLR was in the area of requirements engineering and was proposed by Cleland-Huang et al. [12]. The authors introduced a probabilistic classifier trained on a set of indicator words scraped from websites specifically used to identify non-functional requirements. This technique was later applied to link regulatory codes to project-specific requirements [13] and architectural tactics to source code [14]. Casamayor et al. [15] also proposed an approach for the identification of non-functional requirements built on semi-supervised learning, which used user feedback and Expectation Maximization. Further, Asuncion et al. [16] presented an unsupervised approach that used a modified Latent Dirichlet Allocation algorithm to automatically recover trace links via clustering, which had previously shown promise for automating TLR [17].

In the area of deep learning, Guo et al. [18] suggested a technique to build word embeddings from a domain ontology and generate traceability links using a specialized neural network. Zhao et al. [19] introduced a similar technique that also uses word embeddings, but leverages Learning to Rank to construct trace links based on the similarity of those word embeddings. Other work has also leveraged more complex machine learning models, but still relies on the manual construction of domain ontologies [20]. Unfortunately, domain ontologies are often difficult to construct manually, which poses a very similar situation to building training sets required for supervised learning (i.e., considerable manual effort is required).

Two similar supervised approaches for TLR were proposed in 2018. First, Bella et al. [8] introduced a semi-supervised learning approach with four IR-based similarity metrics as a feature representation. This technique is semi-supervised in that the training set is identified heuristically based on the results of IR. However, compared to active learning, the user is still left to arbitrarily label data based solely on textual similarity. Second, we proposed the TRAIL framework in [9], which is a fully supervised approach that leverages query quality and document statistics features in addition to bidirectional IR rankings as a feature representation. While TRAIL provides an abstracted framework for applying machine learning to TLR with a richer feature representation than the technique proposed by Bella et al., it does not address the issue of acquiring training data. TRAIL rather assumes that training data, representing up to 90% of all the potential links in a system, is already labeled and available and focuses on labeling the remaining 10% of the links. While this is a feasible approach for maintaining already existing traceability data, it is not very conducive for systems where there is fewer traceability information available.

ALCATRAL is different from this body of previous work in several ways. First, it does not require the manual construction of indicator word sets or domain ontologies. Second, it is a supervised technique that leverages bidirectional IR rankings (i.e., semantic similarity), query quality metrics, and simple statistics about each artifact in the potential link to statistically model trace links. Therefore, it is distinct from approaches based on unsupervised learning and word embeddings. Third, it also provides a richer representation than other semi-supervised approaches that have been proposed. Fourth, ALCATRAL improves upon TRAIL by applying active learning to directly address the prohibitive amount of training data required to generate those models. Finally, while the work proposed by Bella et al. addresses the problem of training data tangentially through a semi-supervised approach based on IR-based heuristics, ALCATRAL uses active learning instead.

B. Active Learning

The amount of digital data collected in all aspects of our lives has increased exponentially over the past decade. However, while gathering unlabeled data has become cheap and easy, annotating the data to train machine learning models is still an expensive process in terms of time, labor, and human expertise. Therefore, efforts have been made in the field of machine learning to design techniques aimed at overcoming these obstacles and reducing the amount of labeled training data needed by predictive models. *Active learning* algorithms alleviate this problem by automatically selecting the salient

and exemplar instances from vast amounts of unlabeled data and suggesting these to a human expert for labeling next. In other words, active learning aims at finding the data points that are the most useful for a predictive model in order to learn new information about what distinguishes different classes in the data. Selecting the data points that an expert needs to label using active learning rather than randomly, can prevent a human expert from wasting time annotating data points that do not contribute to a model's predictive performance. This can lead to drastic reductions in the amount of training data needed by predictive models, leading to substantial savings in human annotation effort.

Active learning has been extensively studied by researchers in the machine learning community [21]. The most common variation of active learning is batch mode active learning (BMAL), which exposes the learner to a pool of unlabeled data and then iteratively queries for the labels of a subset of those data points. The most widely used mechanism for determining which data points should be labeled is uncertainty sampling. With this technique, data points for which a model is the most uncertain when predicting class membership are chosen for labeling. That is, with active learning, when a model is confident about its classification of an unlabeled data point it does not need to query for affirmation. For ALCATRAL, model uncertainty is calculated using entropy, which is a common approach [22]. Alternative uncertainty metrics include distance from the classification boundary [23], disagreement between a jury of classifiers [24], variance reduction [25], and the Fisher information matrix [26], [27] among others. Other work has shown that combining multiple selection criteria such as uncertainty, representativeness, and diversity is also a viable strategy for sample selection [28]. More advanced techniques for active learning have also been proposed. For example, Guo and Schuurmans provide a discriminative batch mode approach using Quasi-Newton optimization [29]. Guo also introduced BMAL based on matrix partitioning [30], which is independent of the underlying classification model. Other studies have presented adaptive approaches to automatically batch data based on complexity [31].

To our knowledge, this is the first work that applies active learning specifically to supervised learning for TLR. As such, we have opted to apply a simple implementation based solely on information entropy, which can be built upon in future work. By applying a simple implementation, we can understand the impact that active learning has on automatic TLR via classification and then subsequently measure the impact of improvements. Relying on an approach to active learning that is easily implemented also facilitates adoption by removing unnecessary complexities that provide only minimal performance improvement.

III. Approach

In this section we describe our approach called ALCATRAL (**A**ctive **L**earning for **C**l**A**ssification-based **TRA**ceability **L**inks). Like most recent classification-based approaches to TLR, ALCATRAL is comprised of several components that when implemented together form a predictive model. Unlike these recent approaches, ALCATRAL incorporates *active learning* into TRAIL [9], which represents a novel improvement that reduces the amount of training data required to generate an accurate predictive model. The process ALCATRAL uses to classify potential traceability links is as follows:

1) All *labeled* traceability links are represented as a k-dimensional vector of numerical features that explain relationships between the two artifacts in each link.
2) The feature vectors derived in the previous step are sent through a feature selection routine based on *Pearson correlation*, which removes features that have high pairwise correlation with another feature. The result of this step is a series of p-dimensional vectors that represent each labeled traceability link, where $p \leq k$.
3) *Synthetic Minority Oversampling TEchnique (SMOTE)* is applied to the set of p-dimensional vectors created by the previous step in order to address class imbalance, which occurs because there are many more invalid links than valid links in the overall dataset.
4) Data from the previous step is used to train an initial *Random Forest* classification model, which provides labels and uncertainty scores for the remaining unlabeled links.
5) *Active learning* is applied to the results of the previous step, which produces a list of additional links that are labeled and added to the training set to refine the model.
6) The refined *ALCATRAL* model is used to automatically classify the remaining unlabeled traceability links.

Note that because we are primarily concerned with the application of active learning to reduce the training data requirement of existing supervised classification approaches, we used the same optimal configuration of feature selection (Pearson correlation), rebalancing technique (SMOTE), and machine learning algorithm (Random Forest) we obtained during the extensive evaluation of TRAIL [9] for the implementation of ALCATRAL . By retaining the same configuration, we are able to perform a side-by-side comparison with the state-of-the-art while injecting minimal bias via conflating variables.

A. Representing Potential Traceability Links

Supervised classification-based approaches to TLR consider all of the possible links that could exist between two given sets of software artifacts and predict each to be either valid (i.e., the artifacts are related) or invalid (i.e., the artifacts are not related). Formally, if we have two artifact sets A and B, the approach predicts the validity of each element in the Cartesian product $A \times B$. That is, for each artifact $a \in A$, we predict the validity of the potential link that exists between a and some $b \in B$. Therefore, for the purpose of a prediction model, each potential link between two sets of artifacts is represented by a set of features that are meant to help the machine learning algorithm learn what characterizes valid vs. invalid links. ALCATRAL is implemented with three different types of features based on prior work [9]: IR-rankings, query quality metrics, and artifact statistics.

1) IR-ranking Features: While IR has limitations and does not completely automate TLR [9], it does provide useful information about the semantic similarity between two software artifacts in a potential link. Therefore, we use IR as a set of features that represent the similarity between two artifacts from the perspective of many different IR approaches. This is similar to previous work that has incorporated multiple IR approaches using Learning to Rank [32].

Formally, given two artifact sets A and B, and a possible link between artifacts $a \in A$ and $b \in B$, we capture the strength of this link based on IR using two metrics. First, we use a as a query and the artifacts in B as the corpus. After running a as a query through an IR engine, we capture the rank at which b appears in the list of results as the first metric. Then we repeat the procedure, this time considering b as the query, A as the corpus, and capturing the rank of a in the list of results as the second metric. This gives the model bidirectional information about artifact similarity, as previous work indicated that the search direction directly affects retrieval performance for TLR [33].

For ALCATRAL, we used the same IR approaches initially used by the state-of-the-art classification approach TRAIL [9]: Vector Space Model with TF-IDF weighting and cosine similarity, Okapi BM25, Jensen Shannon, Latent Semantic Analysis, Latent Dirichlet Allocation, Dirichlet language models, and Jelenik-Mercer [34]. We used each to compute the two aforementioned metrics per potential traceability link, resulting in a total of *14 different IR-based features per link.*

2) Query Quality Features: While a significant body of work suggests that IR is a reasonable technique for automating TLR, other work has shown that IR performance is directly related to query quality [35]. In the context of traceability, artifacts with poor textual quality are considered hard-to-trace by IR, as for these artifacts IR provides unreliable results, usually containing many false positives (i.e., an invalid traceability link is erroneously presented as valid). To counteract this phenomena, we employ a series of query quality metrics [35] that are designed to help a model identify the scenario in which two software artifacts appear related based on semantic similarity, but that information is unreliable because of the quality of one or both of the artifacts. We employ all of the features used in previous work in the TRAIL framework [9], for a total of 112 query quality features.

3) Artifact Statistics Features: Previous work in cold-start analytics identified several computationally inexpensive statistical metrics to model trace links [36]. In this work we leverage some of these metrics as features: the number of unique terms in an artifact, the total number of terms in an artifact (including duplicates), and the percentage of overlapping terms between the two artifacts in a candidate link. These features act as proxies for artifact complexity (i.e., longer artifacts are more likely to contain more information) and relatedness (i.e., artifacts that have higher term overlap are more likely to contain similar information). This results in a total of *5 features for each link.*

To summarize, we extracted 131 features for each potential traceability link: 14 IR-based features, 112 query quality metrics, and 5 artifact features. Each feature was normalized to the interval [0,1].

B. Correlation-Based Feature Selection

Some features in the initial 131-feature representation of a traceability link can carry redundant or irrelevant information. This can lead to waisted effort computing unneeded features in addition to overfitting the model. Therefore, in order to identify only the essential features needed for a predictive model, feature selection must be used. While there are numerous approaches available, ALCATRAL performs feature selection using Pearson's correlation as implemented in Weka [1]. This is based on guidance from previous work [9] and supports a one-to-one comparison with TRAIL.

C. The Synthetic Minority Oversampling Technique

Like other classification-based approaches, ALCATRAL considers the set of all possible links between two sets of software artifacts. Generally, in that set there are far fewer valid links than invalid ones. By definition, this leads to a class imbalance problem. If a model is trained without rebalancing the data, it may be difficult to differentiate minority class instances from the majority. Further, the resulting model may not be practically useful, as the model can achieve almost perfect accuracy by näively predicting all links to be invalid. To counteract this phenomena, ALCATRAL uses the Synthetic Minority Oversampling TEchnique (SMOTE) [37]. Using SMOTE, artificial instances of vectors belonging to the minority class are generated by interpolation. That is, new minority class vectors are created *around* existing minority class vectors. This boosts the original minority class data's signal while having minimal impact on the decision boundary.

D. Random Forest

ALCATRAL itself is agnostic to which machine learning classification algorithm is trained on the rebalanced data to perform the prediction. However, to facilitate the comparison with the state-of-the-art, we use Random Forest, as it lead to the best performance for TRAIL [9]. The Random Forest algorithm for classification is an ensemble approach that generates a set (i.e., a "forest") of decision trees and then returns the mode of the predictions made by each tree.

E. Active Learning

Classification-based approaches leverage supervised machine learning to make a prediction of whether a traceability link is valid or invalid; therefore, they require a training set to create the prediction model. In practice, creating this training set is usually delegated to developers who need to inspect a multitude of pairs of software artifacts to determine whether they should be linked or not. Typically, machine learning techniques are evaluated using ten-fold cross validation. Using this technique, it is ensured that each data point in a set is used to test the model exactly once. However, the model is

[1] https://www.cs.waikato.ac.nz/ml/index.html

also trained on 90% of the data and tested on the remaining 10%. Therefore, because the original evaluation of TRAIL used this tactic, it assumed that 90% of the possible links in each software project were already reviewed and labeled by developers. However, as previously discussed, this is not a reasonable assumption for the vast majority of systems, which do not come with complete traceability data.

Unlike TRAIL, which uses a passive learning approach that requires a large set of training data, ALCATRAL uses *active learning* by building upon a *small initial training set* and then *querying* the developer about other traceability links that are difficult for the model to classify. That is, rather than manually labeling the complete dataset, the developer need only label those links the model is most unsure about. After the developer determines whether those links are valid or invalid, they are appended to the initial training set of ALCATRAL. Finally, a new model with reduced uncertainty is trained using this new, enhanced training set. The complete process (also summarized in Figure 2-B) is described as follows:

ALCATRAL first uses a *small initial training set* of labeled traceability links to create a basic classifier, which is expected to have relatively low performance. Then, following a pool-based active learning strategy, the classifier labels the remaining *unlabeled* traceability links (i.e., the pool). When assigning a label, the classifier also provides a confidence score – usually in the interval $[0, 1]$ – which represents how confident the model is in the label it assigned. Using these confidence scores, ALCATRAL samples those links for which there is the most *uncertainty* (i.e., the links for which the model is least confident in its prediction). The expert then inspects each link in that sample and provides the true label for each link. Finally, the newly labeled set of sampled points is appended to the initial training set and an updated model is trained using this new training set. This process can be iteratively repeated until a suitably refined training set is obtained.

To measure the *uncertainty* of a classification using confidence scores, ALCATRAL uses *information entropy*:

$$E(x) = -\sum_{i \in \{0,1\}} P_i(x) \log P_i(x) \qquad (1)$$

Where x is a classified traceability link, i is a label that the classifier can use to classify the link, and $P_i(x)$ is the confidence score of the classifier when it assigns the label i to the link x. A low entropy indicates that the classifier is certain of the label it assigned to link x, whereas a high entropy implies that the classifier does not really know how to classify the link. Figure 1 depicts an example of the sampling process using *entropy* as the measurement of uncertainty. Assume that a classifier tries to assign the label *valid* to an unlabeled traceability link A with a confidence score of 0.9, and assigns the label *invalid* to the same link A with a confidence score of 0.1. In this scenario, the classifier finally classifies the link A as *valid* with high certainty ($E(x) = 0.47$). On the other hand, if the same classifier assigns the *valid* and *invalid* labels to the link B in Figure 1 with confidence scores of 0.4 and 0.6 respectively, it will classify it as *invalid*. Not only is that classification incorrect, it also has high uncertainty ($E(x) = 0.97$). However, if the link B is presented to an expert, she can provide its true label (i.e., valid) by inspecting the artifacts in the link. We can then add link B to the initial training set, retrain the classifier, and obtain an improved decision boundary that now also correctly classifies link C.

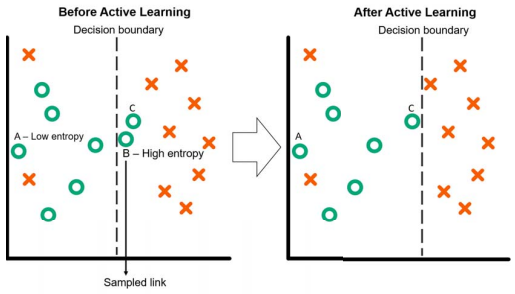

Fig. 1. Pool-based active learning approach used with an entropy-based query strategy. Since links close to the decision boundary are difficult to classify, they are more likely to be sampled and passed to the developer for labeling, in order to obtain their true label.

IV. Design of the Empirical Evaluation

In this paper, our goal is to address the large amount of labeled data required to train predictive models for classification-based approaches to TLR. We performed an empirical study to determine the degree to which ALCATRAL is able to decrease the amount of training data necessary to create accurate predictive models for classifying potential traceability links. This study considers two different ways of applying active learning within ALCATRAL. In the first, we add additional labeled data identified using active learning to a small training set in a single sweep. In the second, we add additional labeled data identified using active learning to the initial training set iteratively, re-labeling the unlabeled dataset and re-calculating the entropy of each of those predicted labels at each iteration. Mind that the systems we use in our study already have the ground truth available, meaning that all the potential traceability links are already labeled as valid or invalid. Therefore, instead of soliciting live help from an expert to label the traceability links we select using active learning, we just consult the ground truth, therefore *simulating* the interaction with an expert developer.

A. Datasets

In our study, for a fair comparison with previous work we use the datasets available in the replication package for TRAIL [9]. We used all 11 traceability datasets available in the replication package. These involve eight different types of artifacts, and are extracted from six different software projects. In total, the datasets have 32,616 possible links between pairs of artifacts, of which 2,600 (7.97%) are valid and 30,016 are invalid (92.03%). By using the same data as TRAIL, we can perform a direct comparison between both approaches to determine the impact that active learning has on performance and amount of training data while holding other conflating

variables constant. Table I shows the breakdown of this dataset by software system, invalid and valid links, and artifact types.

TABLE I
DATASETS USED IN THE EVALUATION

System	Invalid Links	Valid Links	Artifact Types†
eAnci	7091	554 (7.25%)	UC, CC
EasyClinic	1317	93 (6.60%)	UC, CC
EasyClinic	871	69 (7.34%)	ID, CC
EasyClinic	1177	83 (6.59%)	ID, TC
EasyClinic	574	26 (4.33%)	ID, UC
EasyClinic	2757	204 (6.89%)	TC, CC
EasyClinic	1827	63 (3.33%)	TC, UC
eTour	6363	365 (5.43%)	UC, CC
iTrust	1493	58 (3.74%)	UC, CC
MODIS	890	41 (4.40%)	HighR, LowR
SMOS	5656	1044 (15.58%)	UC, CC
Total	**30016**	**2600 (7.97%)**	

† HighR = High-level Requirements, LowR = Low-level Requirements, UC = Use Cases, CC = Code Classes, ID = Interaction Diagrams, TC = Test Cases.

B. Research Questions

Through this empirical study, we seek to answer two research questions:

RQ$_1$: *Can a single application of active learning reduce the amount of training data needed to achieve a performance similar to TRAIL for TLR?* In this research question we seek to understand if applying active learning once can lead to a reduction in the amount of training data needed to achieve a performance similar to TRAIL, which used 90% of all the valid and invalid links in a system as training data. We also replicate TRAIL based on the information provided in the original paper [9], which allows us to then compare the performance of ALCATRAL and TRAIL side-by-side using different amounts of training data. For answering this research question we begin with an initial training set of 10% of all the links in a system and then incrementally add more training data either selected randomly for TRAIL or selected using active learning applied in a single sweep for ALCATRAL.

RQ$_2$: *Does the performance of ALCATRAL improve when active learning is applied iteratively?* For the first research question, we add additional training data for ALCATRAL by using active learning in a single sweep. That is, for any one experiment we compute entropy values, select data points based on the entropy and append them to the training set, then retrain the model only once. Note, however, that entropy is directly related to model uncertainty, which is in turn directly related to model performance. Therefore, intuitively we would expect incremental model improvements to also improve the model's ability to determine its uncertainty about a given prediction. Consequently, we want to explore if this is indeed the case and therefore apply active learning iteratively to answer RQ$_2$. That is, at each iteration (beginning with the same 10% initial training set) we make predictions and compute the associated entropy values, append the 10% of the unlabeled set with the highest entropy to the training set, and retrain the model which is then used as the initial model in the subsequent iteration.

C. Methodology

In all of the experiments discussed below, the same process for splitting the data is used. For each instance of an experiment (i.e., trial), the dataset for each software project is split into three sets: training, unlabeled, and testing using stratified random sampling. In each case, the training and unlabeled sets together make up 90% of the overall data for a project, and the remaining 10% is the test set. The *training set* is a set of labeled links used to train a classification model. The *unlabeled set*, as the name would suggest, is a set of unlabeled links from which additional data is taken for training using active learning. The *testing set* is used to determine a model's performance. Therefore, the training set is used to construct the initial model, the unlabeled set is used to make preliminary predictions, compute entropy for active learning, and select additional training data based on that entropy; then, after retraining including the newly selected data, the testing set is used to determine the performance of that model. Figure 2 illustrates this data splitting process. Note that the testing set is disjoint and solely used to evaluate performance. Further, to address bias potentially introduced by random sampling when splitting the data, we perform 50 trials of each experiment and present aggregated results by averaging the performance of the models across all trials. Note that SMOTE and stratification are applied only for the initial training set. They are not used for retraining models after additional labeled links are added in order to limit the impact of artificial data selection on the results. Further, when referencing $X\%$ of the data, we mean $X\%$ of the total amount of valid and invalid links in a system, for ease of comparison with the original TRAIL experiments which used 90% of the data for training and 10% for testing.

In this work we present the performance of all models in terms of F-Score, a metric that is commonly used to evaluate approaches that support TLR, also used in [9] to compare TRAIL to IR approaches. F-score is the harmonic mean of the precision and recall of an approach, given by:

$$Precision = \frac{|\{valid\ links\} \cap \{links\ classified\ as\ valid\}|}{|\{valid\ links\}|} \tag{2}$$

$$Recall = \frac{|\{valid\ links\} \cap \{links\ classified\ as\ valid\}|}{|\{links\ classified\ as\ valid\}|} \tag{3}$$

$$FScore = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4}$$

F-Score is a reasonable metric for this study because in order for ALCATRAL to be successful, it must balance both precision and recall: automatically recovering the largest number of valid links possible, while also minimizing false positives.

To answer **RQ1**, we train two sets of models. The first is a series of ALCATRAL models trained with increasing amounts

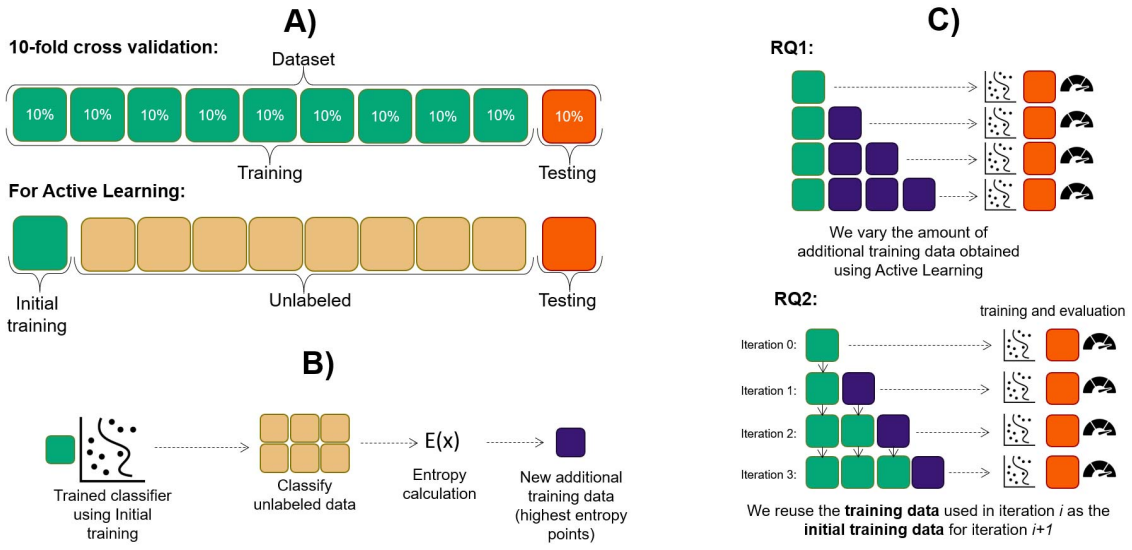

Fig. 2. Data splitting and processing

of data provided by active learning. For these, we begin with a randomly selected initial training set of size 10% chosen with stratified random sampling. For each subsequent experiment we add an additional 10% of the data, retrain the model, and measure its performance with the testing set, until the final experiment where 90% of the data is used as training. The second set of models is a series of TRAIL models that act as a reference for comparison. For these, we train the model with an equivalent amount of data for a corresponding ALCATRAL comparison model using stratified random sampling to select the training data. We perform 50 trials of each experiment, and present the average results.

To fully answer the research question posed, we perform two comparisons. First, we compare ALCATRAL trained with various amounts of data to the TRAIL results provided in [9], which we call the *TRAIL baseline*. This allows us to establish the amount of data at which ALCATRAL achieves parity in performance with the original benchmark. The delta between the 90% of the links used as training data by the TRAIL baseline and the amount of data at which ALCATRAL achieves parity quantifies the improvement achieved by ALCATRAL in terms of reducing the required training data for the best performance. In the second analysis, we compare the performance achieved by ALCATRAL to that of TRAIL using various amounts of training data. This allows us to compare the performance degradation between TRAIL and ALCATRAL as increasingly less data is available for training.

For **RQ2** we begin with an initial training set of 10% of the data selected through stratified random sampling. Then we perform *active learning* by training the model using this data, performing a classification of the unlabeled links based on the trained model, determining the entropy of all unlabeled links and selecting the 10% of the unlabeled links with the highest entropy for labeling next. We then consult the ground truth in order to determine the labels of these previously unlabeled links, and proceed to move them from the unlabeled set to the training set. We then retrain the model with the new training set (including the newly added data points), and continue this process several times, incrementally adding 10% of the remaining unlabeled data with the highest entropy until we have used a full 90% of the overall data (the initial training set plus all of the unlabeled data) to train the model.

As with the previous research questions, we perform 50 trials of this experiment, and present the average results. Further, we provide a comparison to the TRAIL baseline and replicated TRAIL models with equivalent amounts of training data for each ALCATRAL experiment.

We released the scripts used to perform the experiments, the datasets, and the complete set of results in a replication package available at https://bit.ly/2Ip1wK7.

V. Results

A. RQ1

Table II shows the performance of ALCATRAL (ACL) and TRAIL (TRL) models trained with increasing amounts of data in terms of F-Score. Immediately we see a dramatic increase in performance for ALCATRAL models compared to TRAIL models with small amounts of data and, as expected, the difference between the two tapers off as more data is added. Further, while we see improvements in TRAIL models when additional data is available, they do not achieve performance similar to ALCATRAL models until more than 70% of the data is made available for training. Specifically, on average, TRAIL comes within 2 percentage points of performance with an ALCATRAL model trained using 30% of the data only when TRAIL is trained using 70% or more of the data. In general, across all datasets, we see a more than two-fold decrease in the

TABLE II
PERFORMANCE OF ALCATRAL BY AMOUNT OF TRAINING DATA ADDED USING ACTIVE LEARNING AND RANDOM SELECTION

	Initial 10%	Size of the training set																TRAIL baseline as in [9]
		20%		30%		40%		50%		60%		70%		80%		90%		
Dataset		**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	
eAnci	52.2	73.1*	61.4	77.4*	65.9	**77.4***‡	69.0	77.0*	71.7	76.4*	73.6	75.9*	75.1	75.2*	76.7	74.6*	78.0	77.9
eC-CC/UC	45.8	62.4*	53.9	65.5*	57.8	66.5*	60.1	**67.1***†	62.4	67.8*†	63.3	67.6*†	65.0	67.3*†	65.9	67.4†	67.6	67.5
eC-ID/CC	42.1	64.0*	51.6	69.4*	57.9	71.3*	62.7	**71.8***‡	64.6	72.3*†	67.5	71.7*‡	69.4	72.4*†	70.6	72.5†	72.1	72.8
eC-ID/TC	63.2	90.1*	75.6	**93.1***†	81.4	93.6*†	85.1	93.6*†	87.6	93.3*†	89.4	93.7*†	91.3	93.8*‡	92.4	93.6†	93.4	93.4
eC-ID/UC	25.3	50.9*	29.2	52.3*	39.9	54.3*	43.2	**54.7***‡	47.1	55.4*‡	49.0	55.5*‡	51.6	55.6‡	53.9	55.7*‡	57.8	57.6
eC-TC/CC	73.8	94.7*	86.4	**97.7***‡	91.5	98.0*	93.7	98.0*	95.2	97.9*	96.1	98.0*	96.8	98.1*	97.2	98.0*	97.4	97.5
eC-TC/UC	56.2	91.3*	69.4	93.0*	77.1	93.3*	81.9	93.4*	85.8	92.9*	88.1	93.1*	90.9	93.0	92.9	92.9*	94.7	94.5
eTour	42.5	55.0*	49.6	57.0*	52.1	57.5*	54.4	57.9*	56.3	58.2*	57.1	58.3	58.5	58.4*	59.7	58.5*	60.7	60.8
iTrust	30.9	**62.5***‡	47.9	64.1*	53.7	64.5*	57.7	63.7*	59.3	63.3*	60.0	62.7*	60.5	63.0*	61.1	62.3	61.4	60.9
MODIS	32.1	52.0*	42.3	57.2*	48.7	58.6*	53.1	59.8*	56.0	61.2*	59.2	61.6*	60.3	62.6	62.4	62.7*	63.8	64.3
SMOS	47.6	60.2*	56.7	67.0*	62.3	71.4*	66.6	73.8*	70.4	75.2*	73.5	75.9	76.0	76.5*	77.9	76.3*	79.8	79.7
Average	**46.5**	**68.8***	**56.7**	**72.2***	**62.6**	**73.3***	**66.1**	**73.7***	**68.7**	**74.0***	**70.6**	**74.0***	**72.3**	**74.2***	**73.7**	**74.0***	**75.1**	**75.2**

*=there is a significant statistical difference between ALCATRAL and TRAIL trained with the same amount of data. †=there is **no** significant statistical difference between ALCATRAL and the TRAIL baseline [9]. ‡=there is a significant statistical different between ALCATRAL and the TRAIL baseline [9], but the effect size is small or negligible. **Bolded** values show the lowest percentage of training data with which ALCATRAL achieves results that are comparable to the TRAIL baseline [9] (no statistically significant difference or a low or negligible effect size). Underlined values indicate the best result overall for a dataset (for a line).

amount of required training data for ALCATRAL compared to TRAIL to achieve a comparable performance.

It is also important to note that these results show that on average ALCATRAL is within three percentage points of the previously established *TRAIL baseline* (last column in Table II) and our replication of it (second to last column in Table II), even when only using 30% of the data set for training. This both illustrates how duplicative the statistical data contained in the complete dataset is, and indicates that entropy-based active learning is a reasonable means of exploiting that duplicity.

Additionally, one interesting thing to note is that the performance degradation seen for TRAIL models trained with less than 90% of the data is not as severe as one might expect on average, particularly when more than 60% of the data is available for training, in which case the results are within 5% of TRAIL trained with 90% of the data. This serves as a cautionary tale for future evaluations of classification-based approaches: an analysis of the amount of data required to achieve a certain level of performance is extremely important.

Further, while it is interesting to look at the average performance across all of the datasets in consideration, it is also important to consider performance trends for specific datasets. One particularly notable subset of datasets involve EasyClinic: ID/TC, TC/CC, and TC/UC. These are interesting because they each have extremely high F-Score baselines for TRAIL, each above 90%. In each case, ALCATRAL is able to get within $0.2 - 1.5\%$ of that baseline performance using only 30% of the data for training. More than that, in the case of EasyClinic TC/CC ALCATRAL with 30% also surpasses the TRAIL baseline. These results present two important findings. First, the statistical data representing links between these particular artifact sets are particularly duplicative, where a small subset of the data fully expresses the full variability of the overall dataset. Second, these artifact sets provide an opportunity to study the properties of artifacts whose relatedness can more easily be distinguished by a statistical model. Future work should focus on understanding the mathematical properties of these sets in an effort to find ways of making artifact sets more easily automatically traced.

On the other side of the spectrum, ALCATRAL's performance for SMOS and MODIS continues to increase as additional data is added. However, even with 50% of the data available for training, models for these two systems have substantially lower performance than the TRAIL baseline. This indicates that for some datasets, there is insufficient duplicity in the data for active learning to effectively prioritize the links that should be labeled. Further, it indicates that the feature representation presented in the initial TRAIL implementation are insufficient to completely explain the distinction between valid and invalid links to a classifier. As a result, future work should investigate alternative feature representations that could better capture the distinguishing characteristics of valid links.

Finally, in seven of the eleven datasets, ALCATRAL is able to provide performance statistically similar to the TRAIL baseline established with 90% of the data. In those same cases, ALCATRAL provides statistically significantly better results than a TRAIL model trained on an equivalent amount of data. In general, when presented with less than 50% of the total dataset for training, ALCATRAL outperforms TRAIL in every case. Further, when compared to the established TRAIL baseline in [9], ALCATRAL is able to achieve similar performance with significantly less data.

Summary for RQ$_1$. ALCATRAL obtains performance within 3% of the established TRAIL baseline [9] using only a third of the training data. ALCATRAL provides better performance than TRAIL models trained on an equivalent amount of data. ALCATRAL also reduces the training data requirements of TRAIL by at least a half, and in many cases by two-thirds.

B. RQ2

For RQ_2 we employ an iterative approach to active learning in which each subsequent iteration builds upon the model trained by the previous iteration. This approach is based on the idea that an improved model will not only achieve higher performance, but will also provide more reliable entropy calculations. These improved entropy calculations subsequently allow the model to more optimally query an expert for the labels of interesting trace links. Table III shows the results of such an iterative model for each dataset beginning with no additional data, all the way to the 8th iteration, at which the model is trained on the full 90% of the data available in the training and unlabeled sets.

Interestingly, these results indicate that for nine of the eleven datasets, ALCATRAL is able to achieve performance that is statistically similar to the established TRAIL baseline [9] with $\leq 50\%$ of the data used for training. In the case of iTrust, statistical similarity is achieved after the first iteration, requiring only 20% of the overall data. Further, for MODIS and SMOS, the datasets that ALCATRAL struggles the most with when adding additional data in a single sweep, the iterative approach is able to achieve parity with TRAIL using only 50% and 40% of the data respectively for training. This indicates that at least for these two projects that the entropy values computed by the initial model trained on 10% of the data were, in fact, unreliable, which lead to a drastic reduction in efficiency when selecting additional data to be labeled.

Moreover, for eAnci, one of the two datasets that required more than 50% of the data to achieve statistically similar results to the TRAIL baseline, the performance is very close to the baseline for much lower amounts of data, but the results lose statistical significance at iteration 7. Therefore, practically speaking ALCATRAL is likely to provide sufficient performance for smaller training sets, but should be empirically evaluated on a project-by-project basis. This is also important because the artifact types being traced also impact performance as seen in the EasyClinic datasets.

Additionally, Table IV shows performance improvements made by ALCATRAL using an iterative active learning approach over TRAIL for training set sizes. These results clearly show that ALCATRAL provides superior F-Score performance compared to TRAIL particularly for small training sets, which is the most common scenario for software projects in the wild. Overall, ALCATRAL provides a 25% improvement for the smallest training sets, with a minimum and maximum improvement across systems of 11% and 78% respectively. Finally, note that there is still an improvement of more than 10% for most datasets when only 50% of the data is available for training. ALCATRAL and TRAIL begin reaching parity with one another at 60% training sets for most systems. It's important to point out that even though statistical significance between the performance of TRAIL and ALCATRAL exists in certain cases, the effect sizes in those cases are low or negligible. Therefore, TRAIL models do not out perform ALCATRAL models in a practical sense (i.e., large mean differences with statistical significance) even with large amounts of training data available. Therefore, ALCATRAL is able to assist in scenarios in which only limited training data is available and maintains good performance when a larger amount of training data becomes available.

Finally, compared to ALCATRAL models implemented with a single sweep approach to active learning, ALCATRAL with an iterative implementation of active learning provides equivalently strong results with more marked statistical significance. The iterative models achieve parity with the 90% TRAIL baseline [9] earlier and maintain significant differences with TRAIL models trained with less data.

Summary for RQ_2. Compared to single sweep, an iterative approach to active learning in ALCATRAL leads to better performance with statistically significant results over TRAIL trained on equivalent amounts of data. ALCATRAL outperforms TRAIL for small sizes of training data and achieves parity with it when large amounts of training data are available. Therefore, ALCATRAL represents an improvement over TRAIL in the most likely scenario for projects in the wild and maintains similar performance to TRAIL in rare situations where a large amount of training data is available.

VI. Threats to Validity

Construct validity refers to how well the metrics used for evaluation measure the phenomena under study. To mitigate threats to construct validity, we measure the performance of ALCATRAL models using F-Score, which is a standard metric used in the field. By using F-Score we also ensure that we are preferring models with both high recall and precision, which are pragmatic for automatically performing TLR in practice.

Internal validity refers to how well a study insulates changes in the dependent variable from unaccounted for, conflating, independent variables. This study mitigates threats to internal validity in several ways. First, there are many sources of randomization in the experiments presented in this study: splitting the dataset, performing SMOTE, and the random forest classifier itself. In order to mitigate any biases introduced by this randomness, we perform 50 trials of each experiment and present average results across those trials. Further, we have controlled the seeds for both the SMOTE and random forest components of the experiments so that the results can easily be replicated for verification. Second, these experiments are performed on the same datasets used to establish the TRAIL baseline, which is used as a point of comparison for ALCATRAL. Third, because ALCATRAL is an approach that specifically addresses the excessive need for training data using active learning, we match the configuration of TRAIL in each of the comparisons. Using an identical configuration makes a direct comparison with TRAIL possible while minimizing external influences on model performance and highlighting the impact that active learning has on classification-based TLR.

External validity refers to how well the results of a study generalize to other contexts. This study mitigates threats to

TABLE III
PERFORMANCE OF AL-TRAIL WITH ITERATIVE TRAINING

	Initial 10%	Size of the training set																TRAIL baseline as in [9]
		20%		30%		40%		50%		60%		70%		80%		90%		
Dataset		**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	**ACL**	**TRL**	
eAnci	52.2	72.8*	61.4	78.7*	65.9	78.8*	69.0	78.8*	71.7	78.6*	73.6	78.3*	75.1	**78.2***‡	76.7	78.1‡	78.0	77.9
eC-CC/UC	45.8	62.4*	53.9	66.1*	57.8	**67.7***†	60.1	67.3*†	62.4	67.6*†	63.3	67.4*†	65.0	67.1*‡	65.9	67.5†	67.6	67.5
eC-ID/CC	42.1	64.3*	51.6	70.5*	57.9	**72.4***†	62.7	72.3*†	64.6	72.4*†	67.5	72.5*†	69.4	72.6*†	70.6	72.6†	72.1	72.8
eC-ID/TC	63.2	90.0*	75.6	94.1*	81.4	**93.6***‡	85.1	93.6*†	87.6	93.6*†	89.4	93.5*†	91.3	93.4*†	92.4	93.5†	93.4	93.4
eC-ID/UC	25.3	52.0*	29.2	54.9*	39.9	53.7*	43.2	**55.5***‡	47.1	56.6*†	49.0	57.3*†	51.6	56.7*†	53.9	57.2†	57.8	57.6
eC-TC/CC	73.8	95.2*	86.4	**97.6***‡	91.5	97.7*	93.7	97.6*†	95.2	97.6*†	96.1	97.5*†	96.8	97.5*†	97.2	97.4†	97.4	97.5
eC-TC/UC	56.2	91.8*	69.4	**94.9***‡	77.1	95.3*	81.9	95.2*‡	85.8	94.5*†	88.1	94.7*†	90.9	94.8*‡	92.9	94.6†	94.7	94.5
eTour	42.5	56.1*	49.6	58.6*	52.1	59.7*	54.4	60.1*	56.3	**60.5***†	57.1	60.7*†	58.5	61.1*†	59.7	60.6†	60.7	60.8
iTrust	30.9	**62.2***‡	47.9	63.1*	53.7	63.4*	57.7	62.1*	59.3	61.8*‡	60.0	60.7†	60.5	61.2†	61.1	61.3†	61.4	60.9
MODIS	32.1	52.2*	42.3	59.5*	48.7	61.6*	53.1	**62.7***‡	56.0	63.3*‡	59.2	63.6*†	60.3	64.0*†	62.4	64.1†	63.8	64.3
SMOS	47.6	62.9*	56.7	75.3*	62.3	**79.9***‡	66.6	80.0*‡	70.4	79.4*‡	73.5	79.3*	76.0	79.6*†	77.9	79.7†	79.8	79.7
Average	**46.5**	**69.3**	**56.7**	**73.9**	**62.6**	**74.9**	**66.1**	**75.0**	**68.7**	**75.1**	**70.6**	**75.0**	**72.3**	**75.1**	**73.7**	**75.2**	**75.1**	**75.2**

*=there is a significant statistical difference between ALCATRAL and TRAIL trained with the same amount of data. †=there is **no** significant statistical difference between ALCATRAL and the TRAIL baseline [9]. ‡=there is a significant statistical different between ALCATRAL and the TRAIL baseline [9], but the effect size is small or negligible. **Bolded** values show the lowest percentage of training data with which ALCATRAL achieves results that are comparable to the TRAIL baseline [9] (no statistically significant difference or a low or negligible effect size). Underlined values indicate the best result overall for a dataset (for a line).

TABLE IV
PERFORMANCE INCREASE ACHIEVED BY ALCATRAL OVER TRAIL USING DIFFERENT AMOUNTS OF DATA

	Size of the training set							
Dataset	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
eAnci	18.6%	19.4%	14.2%	10.0%	6.9%	4.4%	2.0%	0.2%
eC-CC/UC	15.8%	14.3%	12.5%	7.8%	6.7%	3.7%	1.8%	-0.1%
eC-ID/CC	24.6%	21.8%	15.5%	11.9%	7.2%	4.4%	2.8%	0.7%
eC-ID/TC	19.0%	15.6%	10.0%	6.9%	4.7%	2.4%	1.1%	0.2%
eC-ID/UC	78.0%	37.7%	24.3%	17.9%	15.5%	10.9%	5.2%	-1.0%
eC-TC/CC	10.2%	6.7%	4.2%	2.5%	1.5%	0.7%	0.3%	0.0%
eC-TC/UC	32.3%	23.2%	16.3%	11.0%	7.2%	4.2%	2.0%	0.0%
eTour	13.2%	12.4%	9.7%	6.8%	6.0%	3.7%	2.3%	-0.2%
iTrust	29.9%	17.4%	9.8%	4.8%	2.9%	0.3%	0.1%	-0.2%
MODIS	23.3%	22.1%	16.0%	12.0%	6.9%	5.5%	2.6%	0.4%
SMOS	11.0%	20.8%	20.0%	13.7%	8.1%	4.4%	2.2%	-0.2%
Average	**25.1%**	**19.2%**	**13.9%**	**9.6%**	**6.7%**	**4.1%**	**2.0%**	**0.0%**

external validity by applying the technique to 11 widely used traceability datasets across six different projects spanning eight different types of artifacts. This is not to say the results of this study are completely generalizable, but does suggest that the results are applicable to those types of artifacts and for several systems from different domains. Additionally, we evaluated our approach with the assumption that a developer makes no mistakes at providing a label to the links that are included in the training set. However, mislabeling of links is a possible scenario that can impact ALCATRAL's performance. Future work will focus on estimating the sensitivity of ALCATRAL to mislabeled links.

VII. ACKNOWLEDGMENTS

Sonia Haiduc is supported in part by the National Science Foundation grant 1846142.

VIII. CONCLUSION AND FUTURE WORK

We introduced ALCATRAL, and approach which uses entropy-based active learning as a mechanism to reduce the training data demands of TRAIL, the state-of-the-art supervised machine learning approach to automating TLR. First, we show that ALCATRAL provides models with better performance than training TRAIL on an equivalent amount of training data. Second, we show that using ALCATRAL, models can achieve performance similar to the TRAIL baseline while reducing the amount of training data needed by up to two-thirds. Specifically, we show that ALCATRAL achieves average performance within 3% of TRAIL when only 30% of the traceability data is labeled for training. Finally, we also show that applying AL iteratively provides improved performance over TRAIL with statistical significance in many cases. ALCATRAL with iterative active learning represents an improvement over TRAIL when there is little training data available and maintains similar performance to TRAIL in rare situations where a large amount of training data is available.

Future work will focus on further reducing the need for training data by investigating two research directions. First, we plan to consider smaller increments of additional data than 10%, with particular focus on the iterative approach. Second, we will look at transfer learning, which has been successfully applied to defect prediction [38], [39], as an alternative approach that completely eliminates the need for project-specific training data by training models on data from *similar* projects. By equipping a classification-based approach with a means of completely circumventing the need for project-specific training data, it becomes an even stronger candidate for a fully automated approach to general TLR.

REFERENCES

[1] E. Bouillon, P. Mäder, and I. Philippow, "A survey on usage scenarios for requirements traceability in practice," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2013, pp. 158–173.

[2] P. Mäder and A. Egyed, "Do developers benefit from requirements traceability when evolving and maintaining a software system?" *Empirical Software Engineering*, vol. 20, no. 2, pp. 413–441, 2015.

[3] P. Rempel and P. Mäder, "Preventing defects: The impact of requirements traceability completeness on software quality," *IEEE Transactions on Software Engineering*, 2016.

[4] L. 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*. IEEE, 1997, pp. 2–9.

[5] K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer, "Scenarios in system development: current practice," *IEEE Software*, vol. 15, no. 2, pp. 34–45, 1998.

[6] G. Antoniol, C. Casazza, and A. Cimitile, "Traceability recovery by modeling programmer behavior," in *Proceedings of the Seventh Working Conference on Reverse Engineering*. IEEE, 2000, pp. 240–247.

[7] J. L. de la Vara, M. Borg, K. Wnuk, and L. Moonen, "An industrial survey of safety evidence change impact analysis practice," *IEEE Transactions on Software Engineering*, vol. 42, no. 12, pp. 1095–1117, 2016.

[8] E. E. Bella, M.-P. Gervais, R. Bendraou, L. Wouters, and A. Koudri, "Semi-supervised approach for recovering traceability links in complex systems," in *2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2018, pp. 193–196.

[9] C. Mills, J. Escobar-Avila, and S. Haiduc, "Automatic traceability maintenance via machine learning classification," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 369–380.

[10] M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1565–1616, 2014.

[11] M. Saleem and N. M. Minhas, "Information retrieval based requirement traceability recovery approaches-a systematic literature review," *University of Sindh Journal of Information and Communication Technology*, vol. 2, no. 4, pp. 180–188, 2018.

[12] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements Engineering*, vol. 12, no. 2, pp. 103–120, 2007.

[13] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," in *Proceedings of the International Conference on Software Engineering*, vol. 1, 2010, pp. 155–164.

[14] M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Cinar, "A tactic-centric approach for automating traceability of quality concerns," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2012, pp. 639–649.

[15] A. Casamayor, D. Godoy, and M. Campo, "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," *Information and Software Technology*, vol. 52, no. 4, pp. 436–445, 2010.

[16] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proceedings of the 32nd International Conference on Software Engineering*, vol. 1. IEEE, 2010, pp. 95–104.

[17] C. Duan and J. Cleland-Huang, "Clustering support for automated tracing," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 2007, pp. 244–253.

[18] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 3–14.

[19] T. Zhao, Q. Cao, and Q. Sun, "An improved approach to traceability recovery based on word embeddings," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2017, pp. 81–89.

[20] Z. Li and L. Huang, "Tracing requirements as a problem of machine learning," *International Journal of Software Engineering & Applications*, vol. 9, no. 4, pp. 21–36, 2018.

[21] B. Settles, "Active learning literature survey," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2009.

[22] A. Holub, P. Perona, and M. C. Burl, "Entropy-based active learning for object recognition," in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 2008, pp. 1–8.

[23] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of machine learning research*, vol. 2, no. Nov, pp. 45–66, 2001.

[24] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby, "Selective sampling using the query by committee algorithm," *Machine learning*, vol. 28, no. 2-3, pp. 133–168, 1997.

[25] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," *Journal of artificial intelligence research*, vol. 4, pp. 129–145, 1996.

[26] S. C. Hoi, R. Jin, and M. R. Lyu, "Large-scale text categorization by batch mode active learning," in *Proceedings of the 15th international conference on World Wide Web*. ACM, 2006, pp. 633–642.

[27] S. C. Hoi, R. Jin, J. Zhu, and M. R. Lyu, "Semi-supervised svm batch mode active learning for image retrieval," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008, pp. 1–7.

[28] D. Shen, J. Zhang, J. Su, G. Zhou, and C.-L. Tan, "Multi-criteria-based active learning for named entity recognition," in *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2004, p. 589.

[29] Y. Guo and D. Schuurmans, "Discriminative batch mode active learning," in *Advances in neural information processing systems*, 2008, pp. 593–600.

[30] Y. Guo, "Active instance sampling via matrix partition," in *Advances in Neural Information Processing Systems*, 2010, pp. 802–810.

[31] S. Chakraborty, V. Balasubramanian, and S. Panchanathan, "Adaptive batch mode active learning," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 8, pp. 1747–1760, 2015.

[32] D. Binkley and D. Lawrie, "Learning to rank improves IR in SE," in *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution (ICSME'14)*. IEEE, 2014, pp. 441–445.

[33] C. Mills and S. Haiduc, "The impact of retrieval direction on ir-based traceability link recovery," in *39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track*. IEEE, 2017, pp. 51–54.

[34] C. Zhai and J. Lafferty, "A study of smoothing methods for language models applied to ad hoc information retrieval," in *Proceedings of the 24th annual international ACM SIGIR Conference on Research and Development in Information Retrieval*, 2001, pp. 334–342.

[35] C. Mills, G. Bavota, S. Haiduc, R. Oliveto, A. Marcus, and A. D. Lucia, "Predicting query quality for applications of text retrieval to software engineering tasks," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 26, no. 1, p. 3, 2017.

[36] J. Guo, M. Rahimi, J. Cleland-Huang, A. Rasin, J. H. Hayes, and M. Vierhauser, "Cold-start software analytics," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 142–153.

[37] N. Chawla, K. Bowyer, L. Hall, and P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, Jun. 2002.

[38] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 409–418.

[39] X. Yu, J. Liu, W. Peng, and X. Peng, "Improving cross-company defect prediction with data filtering," *International Journal of Software Engineering and Knowledge Engineering*, vol. 27, no. 09n10, pp. 1427–1438, 2017.