A Tree-Based Approach to Preserve the Privacy of Software Engineering Data and Predictive Models

Yu Fu, A. Güneş Koru[*] , Zhiyuan Chen
Department of Information Systems
UMBC
Baltimore, MD, 21250
{yufu1|gkoru|zhchen}@umbc.edu

Khaled El Emam
University of Ottawa
Faculty of Medicine and the School of
Information Technology and Engineering
Ottawa, CA
kelemam@uottawa.ca

ABSTRACT

In empirical disciplines, data sharing leads to verifiable research and facilitates future research studies. Recent efforts of the PROMISE community contributed to data sharing and reproducible research in software engineering. However, an important portion of data used in empirical software engineering research still remains classified. This situation is unlikely to change because many companies, governments, and defense organizations will be always hesitant to share their project data such as, effort and defect data, due to various confidentiality, privacy, and security concerns. In this paper, we present, demonstrate, and evaluate a novel tree-based data perturbation approach. This approach does not only preserve privacy effectively, but it also preserves the predictive patterns in the original data set. Consequently, the empirical software engineering researchers will have access to another category of data sets, *transformed data sets*, which will increase the verifiability of research results and facilitate the future research studies in this area. Our approach can be immediately useful to many researchers and organizations who are willing to share their software engineering data but cannot do so due to privacy concerns.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics/Measurement—*complexity measures, performance measures*; M.13.3 [**Services Computing**]: Security and Privacy in Services—*privacy management in data transformation, privacy management in data dissemination*; K.4.1 [**Computing Milieux**]: Computers and Society—*Privacy*

[*]Corresponding Author.

1. INTRODUCTION

An important characteristic of the PROMISE effort has been its emphasis on developing an infrastructure where researchers share not only their results and papers, but also the data used in their studies. In addition, PROMISE conference encourages researchers to use the data sets available in the PROMISE repository. Benefits of data sharing have been observed and expressed in other scientific domains, e.g., in bioinformatics [11]. The UCI's machine learning repository of public data sets have been acknowledged by software engineering researchers as an exemplary effort that needs to be adopted [8]. Reporting not only the research results but also the underlying research data increases the verifiability of the research results and facilitates further research studies exploring similar or related questions.

Despite its potential benefits, both obtaining and sharing data pertaining to software development efforts has been a major challenge for software engineering researchers [8, 9]. A recent systematic review of software defect prediction studies found that, in spite of the recent efforts of the PROMISE repository, only 31% of the studies made their data public [7]. An important reason is that companies consider the potential use or misuse of software development data a major risk that can damage their reputation or cause business disadvantages. Defense and government organizations have generally similar concerns; in addition, they might even have stricter policies to assure confidentiality. Therefore, facilitating data sharing is still a major challenge in the field of empirical software engineering. We are in need of techniques that can transform software engineering data to preserve the privacy of software engineering professionals, projects, and organizations.

Note that, in preserving privacy, removing the identifiers in a data set such as names, addresses, personal id numbers, etc., is insufficient. Even without looking at such identifiers, inspectors can examine the values of various other variables and identify the owner of a data point (e.g., a person, project, or a module) by also using other information available to them. Alternatively, they might be able to narrow down the possible owners of a data point to a small set, which significantly increases the chances for successful identification. Consider the original COCOMO data set in Table 1 (in the Appendix) where each data point corresponding to a project includes a product size value in Kilo–lines-of-code (*loc*). If an inspector knows which products were the largest products and some of the project characteristics for those largest products that are coded in the data set, such as, the use of tools (*tool*), database size (*data*), time and memory constraints (*time* and *stor*), it will be easier to identify the individual projects among those with the largest products. Then, this identification can reveal how personnel variables were scored in the identified projects, for example, analysts capability (*acap*) and programmers capability (*pcap*). Therefore, preserving privacy through *data perturbation* becomes necessary, which can increase the chances for data sharing by alleviating some of the confidentiality concerns.

For data perturbation, a number of techniques such as generalization [2, 10, 17] or adding random noises [1] could be used to anonymize software engineering data. However, those techniques were designed with other utility concerns in mind, such as anonymizing patient records in hospital databases to protect patient privacy but still being able to extract and use patient data for reporting purposes. They fail to effectively preserve the predictive patterns in the original data. Preserving such patterns is necessary if data will be used for research purposes, especially when building predictive models, because the patterns indicate how certain predictor variables relate to the outcome variable. For example, consider the researchers building statistical models to predict project effort. As we demonstrate in this paper, if they use the existing data perturbation techniques mainly designed to protect privacy, the patterns for project effort observed from the perturbed data will be lost, leading to low prediction performance.

Preserving both privacy and predictive patterns is an important but, at the same time, a challenging problem because preserving privacy often requires distortion to the data whereas preserving patterns requires the opposite. As a starting point, in this paper we focus on perturbing data by preserving a specific type of patterns, those formed by the rules in regression trees. The use of regression trees in software engineering research can be seen in [22, 27]. Their use in this study is limited to data perturbation purposes. Regression trees do not make assumptions about the functional forms of the relationships between the response and predictor variables. They create a set of rules that partition the data into subsets which are as homogeneous as possible in terms of the response variable values. We observed that the rules formed during this process enable us to perturb data values in certain value ranges without changing the patterns observed in the original data. Based on this observation, we developed a novel tree-based data perturbation technique that we call *Pattern Preserving Tree-based* algorithm, or **PPT** for short.

PPT receives the original data and a regression tree generated using the original data from the researchers who would like to transform their data and share it (called *producers*). Then, it perturbs the original data considering the patterns existing in the original tree. Unlike existing data perturbation techniques, PPT takes special care to preserve those patterns during data perturbation. Finally, it produces a new regression tree from the perturbed data, which can be published along with the perturbed data to other researchers (called *readers*). Indeed the patterns are preserved to the extent that, except showing normalized variables, the new regression tree looks exactly the same as the original one. To our best knowledge, there is not any other available technique or tool achieving this goal. Consequently, other researchers can use the perturbed data effectively to verify earlier results or to explore further research questions.

To be clear, regression trees are only used for data perturbation purposes in this study, not to make predictions. That is, we are not suggesting that researchers should always use regression trees for their modeling purposes. We suggest that researchers use the modeling techniques best suited to their particular research questions. Furthermore, we encourage them to share their original data sets because any data perturbation method will result in some degree of information loss. However, when sharing the original data is impossible, under certain plausible situations, PPT can be used to *transform* the research data. Currently, the only alternative to PPT is not to publish any data.

To summarize, this study mainly addresses and facilitates the following two challenge areas in software engineering research:

- *Verifiable and repeatable research*: The studies that cannot publish their actual data can publish the transformed data by using PPT. Therefore, other researchers can repeat the analysis employed in the original study by using the transformed data in order to verify and validate the original results. For example, consider some researchers in a company who collected project size and cost drivers data to create a COCOMO model [3]. Due to the proprietary nature of the data, they could use PPT and transform it in order to be able to publish it. Other researchers building the COCOMO model with the PPT-transformed data would obtain a reasonable prediction performance.

- *Exploration of further research questions*: In the long-run, an accumulation of such transformed data sets can enrich the repertoire of the data sets available to researchers. This will allow researchers and practitioners to build prediction models while exploring similar or relevant questions by also giving them more insight about the patterns observed in the proprietary and/or confidential data sets.

In the rest of the paper, we start by introducing the related work. Then, we introduce the methods used in the study; we explain the overall process of perturbing the COCOMO data set; we discuss the algorithm for PPT in detail; we demonstrate the tradeoff between privacy and prediction performance when PPT is used; and, we explain how we produced the final perturbed COCOMO data. Then, in our discussion section, we mention where PPT is applicable and effective. Finally, we conclude the paper. Following the tradition of the PROMISE effort, both the original and perturbed data sets are included in the Appendix, and they will be available at the PROMISE repository. In addition, an R package including the programs that implement the PPT algorithm will be made available at the PROMISE repository.

2. RELATED WORK

Recently, privacy has also become an important issue for the software engineering community, in addition to the traditional quality attributes such as reliability, usability, performance, etc. In a recent article, Spiekermann and Cranor [23] discuss in length how to engineer privacy in information technology products. Their discussion mostly focuses on understanding the privacy requirements of users and system stakeholders, and how to design software systems to meet those privacy requirements.

A similar study was conducted by Canfora and Visaggio [5], which appeared in the International Symposium on Empirical Software Engineering and Measurement (ESEM). The researchers stressed the potentials of using anonymization techniques in engineering software systems, and demonstrated the use of those techniques using a medical database. They pointed to the trade-off between data quality and anonymization, and stated that this trade-off should be managed for each application. The authors' more recent study [6] reported similar results.

Different from [5, 6, 23], in this research, our goal is to facilitate the use and sharing of data collected for research purposes. Such data could be related to not only users or stakeholders of software systems, but also related to software developers, managers, projects, processes, development environments, and organizations.

Cukic and Ma [8] mention that researchers and companies are sensitive when it comes to sharing software development data despite the advances in data anonymization techniques. However, it is worth to note that the most advanced techniques in data anonymization did not focus on preserving the predictive patterns existing in original data effectively, which might have prevented their adoption in software engineering research.

Nevertheless, there has been a rich body of work on protecting privacy in the database and data mining fields. The existing work can be divided into three categories based on the target of protection: personal identity (called data anonymization), sensitive attribute values, and sensitive patterns. The research in data anonymization is based on two privacy protection models: K-anonymity and L-diversity. The K-anonymity model [24] protects the data against linking attacks. The L-diversity model [20] was proposed as a complement to K-anonymity. L-diversity further protects the data against elimination attack (when attackers can exclude certain values). Many studies have been conducted on how to implement these two models. A widely used approach is generalization, i.e., replacing values of attributes with more general values [2, 10, 17].

Research on protecting sensitive values is also called privacy-preserving data mining because its goal is to preserve privacy and at the same time allow data mining on the modified data. A survey of the existing methods can be found in [30]. The most well known method in this field is random perturbation, which adds some random noise to sensitive attribute values [1]. However, Kargupta et al. [13] showed that random perturbation method is subjected to attacks using correlations of data. A tree-based approach perturbation approach, K-D tree [19] was also proposed. The basic idea is to divide data into groups using K-D tree, and replace values of sensitive attributes with the average of their values within a group.

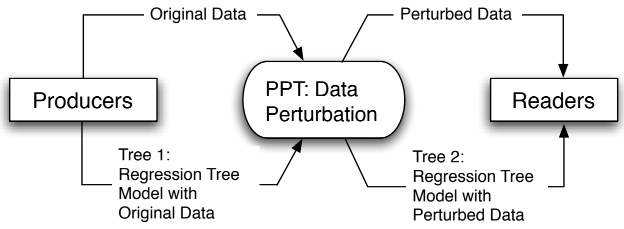
Original Data
Perturbed Data
Producers
PPT: Data Perturbation
Readers
Tree 1: Regression Tree Model with Original Data
Tree 2: Regression Tree Model with Perturbed Data

Figure 1: Using PPT: Original data is normalized and then perturbed heavily. Except showing normalized values, Tree 2 is the same as Tree 1 preserving predictive patterns. The perturbed data can be used to build prediction models by other researchers.

The technique proposed in this paper can preserve the predictive patterns formed by the regression tree models. This is an important point while sharing data for research purposes, however, it has not been the essential focus in the area of privacy preserving data mining. Some studies mentioned that the prediction performance did not degrade much when the perturbed data was used (e.g., [1]). However, the authors did not focus on preserving the patterns in the data. Furthermore, their observation is not guaranteed to apply to all data perturbation or transformation schemes. Indeed, some studies (e.g., [5, 6]) mentioned the loss in the prediction performance as the degree of anonymization increases. To our best knowledge, none of the previous studies could achieve preserving both privacy and predictive patterns at a reasonable level and in a controllable manner as done in this study.

3. METHODS

Figure 1 shows how PPT is used. The producer researchers obtain a regression tree from the original data, represented as Tree 1 in Figure 1. Then, they give Tree 1 and the original data to PPT. PPT generates perturbed data by preserving the patterns in Tree 1. The perturbed data can be published to readers along with the regression tree PPT generated from the perturbed data, Tree 2 in Figure 1. Tree 1 and Tree 2 have the same patterns except that Tree 2 shows the normalized data. Note that, readers can reproduce Tree 2 from perturbed data; they can also use the perturbed data to rebuild the predictive models in order to validate the original research results or to explore further research questions.

To demonstrate the use of PPT, we perturbed a publicly available data set used to predict project effort by Boehm et al. [3]. In the rest of this section, we first discuss the overall process of perturbing the COCOMO data. Then, we present some background and key observations for regression trees in Section 3.2 which led to the development of PPT algorithm. In Section 3.3, we present the algorithm for PPT and explain the perturbation method in detail. Then, we discuss the trade-offs between privacy and prediction performance observed while using PPT. Lastly, we explain how we produced the final perturbed COCOMO data.

3.1 Perturbation of COCOMO Data

The COCOMO data set we used is available at the PROMISE repository [21] (called COC81). There are altogether 63 data points (records) in this data set, each representing a software project. All variables in this data set have numerical values. The response variable is effort in person-months (*actual*), and there are 16 predictor variables including the project size in Kilo-lines of code (*loc*).

The COCOMO model can be written as:

$$Effort = a * Size^b * \prod_i Cost\ Driver_i \qquad (1)$$

This model suggests a linear relationship between the logarithmic transform of $Effort$ and the logarithmic transform of $Size$ and the cost drivers. That is,

$$\ln(Effort) = \ln(a) + b * \ln(Size) + \sum_i \ln(Cost\ Driver_i). \qquad (2)$$

Therefore, we performed the following steps:

1. We built a linear ordinary–least-squares (OLS) regression model [12] to predict $\ln(Effort)$ using the logarithmic transforms of the predictor variables, $Size$, and the cost drivers in the original data. Table 1, which is in the Appendix, shows the original COCOMO data set in the log-transformed format[1].

2. We obtained a regression tree from the log-transformed COCOMO data. Then, we gave this tree to PPT along with the log-transformed data (as depicted in Figure 1) as inputs. To build regression trees, we used the *rpart* package [26] available in the R statistical environment [25]. PPT provided us with a perturbed data set.

3. On the perturbed data set, we built another linear OLS regression model and compared its prediction performance with that of the original model by using the adjusted R^2 values obtained for both models. During this process, we always used bootstrapping and shrunk the models to correct for optimism in the adjusted R^2 values. To quantify the level of privacy for the perturbed data set, we use the confidence interval measure [1] that measures the average privacy. This metric measures how closely the original value of a perturbed variable can be estimated. If a perturbed attribute x can be estimated with $c\%$ confidence in the interval $[x_1, x_2]$, then the interval $x_2 - x_1$ indicates the degree of privacy. The larger the interval, the better the privacy protection. For example, suppose the interval for a variable is 0.4, and the perturbed value of the variable is 0.5, then the original value may lie anywhere in 0.3 to 0.7 (i.e., 0.5-0.2 to 0.5+0.2). 95% confidence interval was used in our experiments. Since different variables may fall in different ranges, the interval is also normalized to $\frac{x_2-x_1}{x_U-x_L}$ where x_U is the maximal value of x and x_L is the minimal value. We compute this measure over all perturbed variables and report the average.

4. We repeated Steps 2 and 3 many times by building regression trees of different sizes and giving them to PPT. We achieved this by manipulating the *complexity parameter (cp)* of the *rpart* function. As we demonstrate later, manipulating *cp* also changes privacy level and prediction performance obtained from the PPT-perturbed data. Finally, we chose a *cp* value that provided a good trade-off between privacy and prediction performance, and we repeated Step 2 to obtain the final perturbed data set, which is shown in Table 2 of the Appendix.

[1]More precisely, log1p function in the R statistical environment [25] was used. This function adds 1 before taking the natural logarithm in order to avoid the problem of taking the logarithm of zero, which is undefined.

3.2 Background and Key Observations for Regression Trees

Regression trees are often used to predict the value of a numerical response variable such as person-months, change count, number of defects, etc. [14, 15, 16, 22, 27, 28, 29]. Typically, a regression tree, T, is obtained by recursively partitioning all of the data points in a data set. The root node of the tree includes *all* of the data points in the data set. For each partitioning step, a predictor variable and a *split* value (also called *cut-off* value) is chosen such that the partitioning will minimize the total deviance of the response variable in the child nodes. In other words, the maximum deviance reduction possible is obtained in each split. The selection of a split variable-value pair is performed automatically using the available statistical analysis packages because it requires trying many candidate predictors and split values [4]. Each node also includes the mean of response variable as the predicted value.

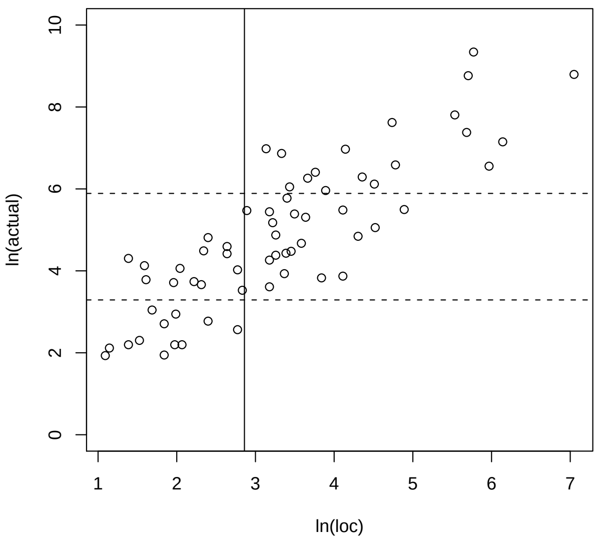

Figure 2: The split at top node for COCOMO data.

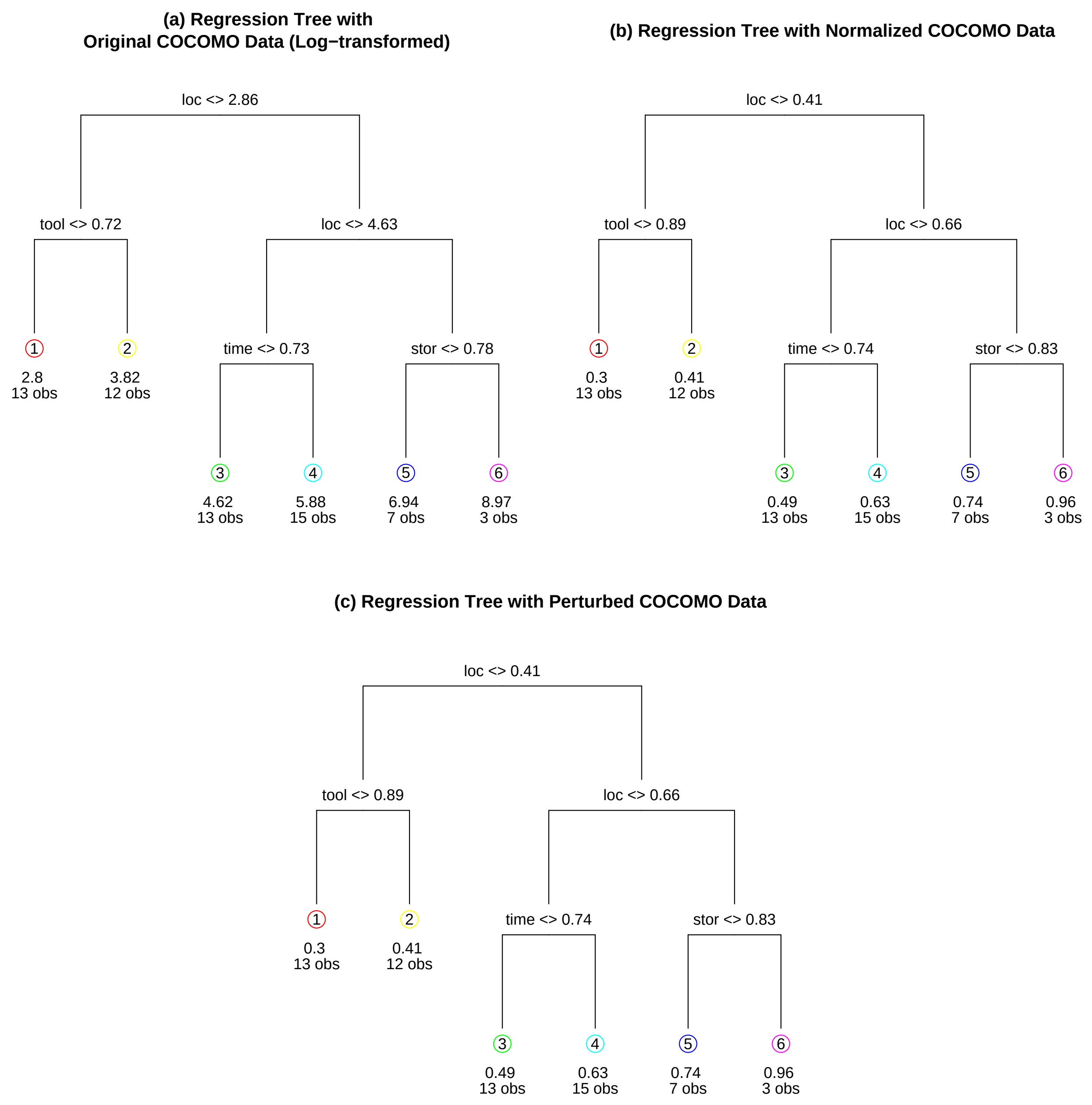

Figure 3: Regression trees generated using original and perturbed data.

For example, Figure 3-(a) shows the regression tree generated from the log-transformed original COCOMO data in Table 1. Each node in the tree contains the split variable and splitting value. For example, in the top split, those with $\ln(loc) \leq 2.86$ go to the left child and those with $\ln(loc) > 2.86$ go to the right child. In each leaf node, the predicted value and the number of records in that leaf are shown in Figure 3.

Figure 2 illustrates how the split value for the top node is found. The tree building algorithm considers all possible split values for each predictor variable. The split that minimizes the deviance will be selected. In Figure 2, the split at $\ln(loc) = 2.86$ minimizes the deviance on response variable (the effort) because the effort values of projects to the left of the split value are mostly quite small while the effort values of projects to the right are mostly quite large. Thus it is selected as the split value. The split value is selected as the midpoint of *boundary values* in the left and right child. The boundary value in the left child is the maximal $\ln(loc)$ value in the left child (2.83 in this case) and boundary in the right child is the minimal $\ln(loc)$ value in the right child (2.89 in this case).

To ensure that the patterns in the original regression tree will be preserved using the perturbed data, we use the following two key observations:

Theorem 1. *If each variable column (also called variable vector) is divided by a constant, the regression tree will remain unchanged except that split values and predicted values in the tree are also divided by that constant.*

The proof of this theorem is quite straightforward because the deviance after the division equals the original deviance divided by square of the constant. Thus, for any two splits, suppose that the deviance of split one is less than that of split two, the same occurs after the division. This will ensure that the same split variable and the corresponding normalized value will be selected at each step in the regression tree building algorithm. Thus, the structure of the regression tree generated from the perturbed data will be exactly the same as the tree generated from the original data.

In this paper, we assume that each variable has a non-negative numerical value. This theorem indicates that we can normalize each variable by dividing the maximal value of that variable, without changing the regression tree. Figure 3-(b) shows the tree generated from the normalized data. This tree is the same as the tree generated from the original data, Figure 3-(a), except that split values and predicted values in the tree are normalized. This step also brings some privacy protection because unless someone knows the maximal value for a variable, one can not discover the actual value of that variable easily. However, note that, this step is just the start of the major perturbation performed by PPT.

Theorem 2. *If data perturbation preserves the order of values for each predictor variable and leaves the response variable value unchanged, the regression tree generated from the perturbed data will be the same as the tree generated from the original data, except that the split values need to be computed from the perturbed boundary values.*

Example 1: The key observation is that the deviance of a certain split only depends on the two sets of response variable values in the two child nodes generated by the split. For example, consider the top node in Figure 3-(a). The split at $\ln(loc) = 2.86$ will generate two child nodes: those $\ln(loc) \leq 2.86$ and those with $\ln(loc) > 2.86$. The deviance is determined by the set of effort values in the two child nodes.

Now, we modify the $\ln(loc)$ values, but make sure the modified values preserve the order in the original data. An important observation is that the modified $\ln(loc)$ values of projects in the left child are still less than the modified $\ln(loc)$ values of projects in the right child due to the order preserving property. Thus each project in the modified data still goes to the same child as in the original data.

For example, suppose that there are two projects, one with $\ln(loc) = 2.83$, and the other with $\ln(loc) = 2.89$. Thus the first project belongs to the left child and the second project belongs to the right child. Now, suppose that the modified $\ln(loc)$ values for these two projects are 2.73 and 2.99, respectively. The first project still belongs to the left child and the second project still belongs to the right child.

The order preserving property ensures that each split will generate the same two sets of projects as in original data. Since we do not modify the response variable values, the two sets of response variables do not change. Hence the deviance after the split also does not change. This will ensure that the tree building algorithm will generate the same split at each step as in the original data, and the same tree will be generated over the perturbed data. The only difference now is the split value needs to be set to the new midpoints between the boundary values in the modified data. For instance, the split value in the above example equals the midpoint of perturbed boundary values (2.73 and 2.99).

3.3 PPT Data Perturbation Algorithm

Figure 4 shows the Pattern Preserving Tree-Based data perturbation algorithm. At Step 1, the algorithm normalizes all variables to take values in the range $[0, 1]$ by dividing each value of a variable in a record by the maximal value of that variable in the data set. As shown in Theorem 1, this step will not change the tree model created from the data.

In Steps 2 to 5, the algorithm computes the minimal and maximal values of each child node in T. For example, in Figure 3-(b), the minimal $\ln(tool)$ value for the left most leaf node (with a circle 1) is 0.5445, and the maximal value is 0.8595. At step 6, these minimal and maximal values will divide the range of each split variable into intervals, where these minimal and maximal values are boundaries of intervals. For example, there is only one node in Figure 3-(b) that have $\ln(tool)$ as split variable. Thus there will be 4 minimal and maximal values (2 for each child node). These values will divide the values of $\ln(tool)$ into 3 intervals: [0.5445, 0.8595], [0.8595, 0.92], and [0.92, 1.0].

At Step 8, each value is replaced with the mean of the interval. For example, any $\ln(tool)$ value in the range of [0.5445, 0.8595] will be replaced with the mean 0.8395. Note that this step will not modify the order of values. For example, if v_1 belongs to [0.5445, 0.8595] and v_2 belongs to [0.92, 1.0], then $v_1 < v_2$. Step 8 will transform v_1 and v_2 to the mean of values in these two intervals, respectively. Note that the mean of values in the first interval must be less than the mean of values in the second interval. For example, the mean in the first interval is 0.8395, which is less than the mean in the second interval, 0.932. Thus the perturbed value of v_1 is also less than the perturbed value of v_2. Since the order is preserved in the perturbed data by Theorem 2, the regression tree generated from the perturbed data will be the same as the regression tree generated from the original data, except that the split values need to be computed from perturbed boundary values. Finally, Steps 9 to 13 will adjust the boundaries of intervals to ensure that the split values in the tree generated from perturbed data will be the same as the split values in the original tree.

We use an example to illustrate how to achieve this. Consider the left most node in the second level of the tree in Figure 3 (b) (the node with condition "$\ln(tool)$ <> 0.89"). The maximal value in the left child equals 0.8595 and the minimal value in the right child equals 0.92. The original split

```
PPT($D$,$T$) where $D$ is the original data and the $T$ is the regression-tree
model whose patterns need to be preserved
1.  Normalize each variable $V_i$ in $D$ by dividing value of $V_i$ by the maximal value of $V_i$.
2.  For each node $N_i$ in $T$
3.       Suppose $V_i$ is the split variable, compute the minimal and maximal of $V_i$ in its left child.
4.       Do the same for the right child.
5.  end for
6.  For each predictor variable $V_i$ in $D$
7.       Use the minimal and maximal values recorded in step 4 and 5 to divide $V_i$ into intervals.
8.       For each interval, replace every value in the interval with the mean in the interval.
9.       For each node in $T$ that has condition $V_i \leq v$ and $V_i > v$
10.          Let $v_{max1}$ be the maximal $V_i$ value in the left child
             and $v_{min2}$ be the minimal $V_i$ in the right child.
             Let $v'_{max1}$ and $v'_{min2}$ be the current value of $v_{max1}$ and $v_{min2}$ set at step 9, respectively.
11.          Compute $\delta = \min\{v_{max1} - v'_{max1}, v'_{min2} - v_{min2}\}$.
12.          Replace $v_{max1}$ and $v_{min2}$ with $v_{max1} - \delta$ and $v_{min2} + \delta$, respectively.
13.       end for
14.  end for
```

Figure 4: Pattern-Preserving Tree-Based Algorithm

value 0.88975 is the midpoint of these two values. At Step 8, these two values are set to 0.8395 (the mean of values in $[0.5445, 0.8595]$) and 0.932 (the mean of values in $[0.92, 1.0]$), respectively. In order to make sure the split point is the same, we need to ensure that the midpoint (average) of perturbed boundary values equals the old split value 0.88975.

Figure 5 illustrates how this is done. Note that the split value in the original tree equals to the average of the original boundaries (0.8595 and 0.92). We can decrease the value of 0.8595 (the original left boundary) and increase the value of 0.92 (the original right boundary) by the same amount δ. This will not change the split value because the average of $0.8595 - \delta$ and $0.92 + \delta$ equals the average of 0.8595 and 0.92. In the mean time, the perturbed value for 0.8595 must be greater than or equal to the mean in the left interval (0.8395) because all other "ln($tool$)" values in the left child has been perturbed to 0.8395 and we need to preserve the order. Similarly, the perturbed value for 0.92 must be less than or equal to the mean in the right interval (0.932). Thus step 12 computes the maximal possible δ, which equals 0.012 (the min of $0.8595 - 0.8395$ and $0.932 - 0.92$). Finally 0.8595 is set to 0.8475 and 0.92 is set to 0.932. The midpoint is still 0.88975.

Our approach has the following favorable characteristics:

1. The regression tree model generated using the perturbed data is the same as the model generated from the original data. This can be proved as follows. First, by Theorem 1, the normalization step (Step 1) preserves the regression tree. Second, subsequent steps of the PPT algorithm do not change the order of values. Thus, by Theorem 2, the regression tree is preserved except for split values. Finally, Steps 9 to 13 ensure that the split values in the tree are also preserved.

2. The data values have been perturbed heavily as follows. The values of a predictor variable that appears in the regression tree are set to mean of intervals generated in Step 8. The values of a predictor variable that does not appear in the regression tree (thus not too useful for prediction) are set to the mean of that variable. The values of the response variable are also normalized such that it will be difficult to know the exact values without knowing the maximal value in the data set. As a result, privacy is greatly preserved.

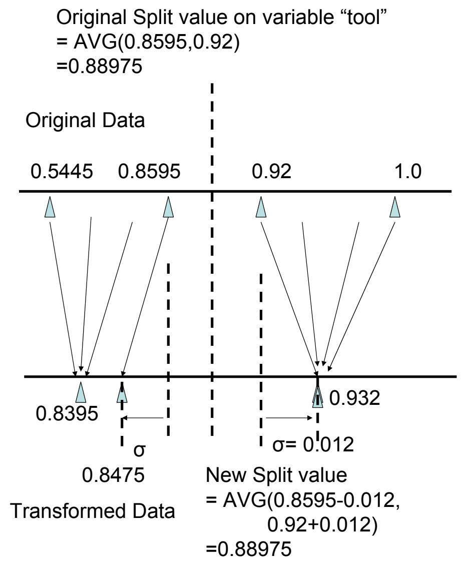

Figure 5: Example of preserving split values using ln($tool$).

4. RESULTS

We perturbed the log-transformed COCOMO data by executing the PPT algorithm with different sizes of regression trees (denoted as T in Figure 4). While doing so, we observed how the privacy and prediction performance changed.

Figure 6 shows the tradeoff between privacy and prediction performance. Using the original log-transformed data (see Table 2), the adjusted R^2 value obtained from the OLS regression model (based on the Equation 2) was 0.92. This value is shown as a horizontal line in Figure 6. The complexity parameter (*cp*) determines the decrease needed in the overall lack of fit in order to attempt a split while building regression trees. The smaller *cp* values typically result in greater numbers of splits and, therefore, larger trees. The larger *cp* values typically result in smaller number of splits and, therefore, smaller trees. We observed that the smaller *cp* values resulted in higher adjusted R^2 obtained from the perturbed data, however, less privacy. The greater *cp* values resulted in lower adjusted R^2 obtained from the perturbed data, however, higher privacy.

The tradeoff shown in Figure 6 is a typical and expected one. Smaller trees usually have more data points per leaf compared to larger trees; therefore, PPT can perturb the data points in those leaves more heavily using the algorithm shown in Figure 4. On the other hand, the larger trees have usually more rules preserving the patterns for the relationships between the outcome and predictor variables better and resulting in higher adjusted R^2 values.

For any data set that needs to be transformed, the producers of the data set should examine the tradeoff between privacy and prediction performance before making a final decision about the input regression tree. Note that the *rpart* function in the *R* statistical environment allows some other parameters to be manipulated too, such as minimum leaf size to attempt a split (*minsplit*) and minimum bucket size (*minbucket*). After examining Figure 6, we decided that $cp = 0.025$ (the vertical line in Figure 6) was a good value to provide a balance between privacy and prediction performance.

Finally, we produced a regression tree with $cp = 0.025$ and used it to produce the final perturbed data set (with default *minbucket* and *minsplit* values). This perturbed data set is shown in Table 2 in the Appendix. The regression tree used for final perturbation is the one shown in Figure 3-a with the original log-transformed COCOMO data. As seen in Figure 3-b, the exact same tree structure is obtained when the log-transformed data is normalized. Furthermore, as seen in Figure 3-c, even after the data is perturbed using the PPT algorithm, the tree structure is preserved because PPT pays particular attention to preserving the pattern observed in the regression tree obtained from the original data set. With $cp = 0.025$, the adjusted R^2 obtained from the simple linear regression model (based on the Equation 2) was 0.82.

It is worth to note that, we also experimented with the existing privacy preserving data transformation approaches in order to understand whether they could be used for our research objective, that is, preserving privacy and prediction performance at the same time. The K-D tree approach [18, 19] is the state-of-the-art tree-based data perturbation approach in the field of privacy preserving data mining. The average privacy we obtained from our K-D tree experiments was around 0.9[2], which was slightly above the highest privacy that can be obtained from the PPT approach. However, compared to PPT, the K-D tree approach was very ineffective when it came to preserving the predictive patterns. With different settings of K-D trees, we were able to obtain, at most, an adjusted R^2 value around 0.30 for the OLS models generated from the K-D tree perturbed data. We also observed that, since K-D tree approach was not created to preserve the predictive patterns, the perturbed data changed greatly when we used different settings to create K-D trees, some of which produced very low adjusted R^2 values.

To summarize, as we demonstrated, PPT did not only produce the exact same regression tree showing the same patterns, it also resulted in a reasonable prediction performance when we generated the OLS models from the perturbed data based on the COCOMO model. Therefore, PPT successfully met our research objective of preserving privacy while preserving the utility of the perturbed data when used for research purposes.

5. DISCUSSION

We developed a tree-based perturbation approach because regression trees have nice properties that give us some intervals in which it is possible to perturb data flexibly without causing changes in the patterns observed for the original data.

Naturally, the quality of the conclusions that can be drawn from the perturbed data is also related to the internal validity and prediction performance of the original model obtained from the original data set. For example, in this research, we used the COCOMO model pointing to a linear relationship between the logarithm of effort and the logarithms of the predictor variables (e.g. size). Since the OLS model based on Equation 2 gives an adjusted R^2 value of 0.92 (after correcting for optimism) for the original data set, the OLS model obtained from the perturbed data leads to useful models with high prediction performance. However, when the prediction performance obtained from the original model is very low or when there are known internal validity threats, data perturbation might not lead to useful data sets. Otherwise, using PPT and publishing perturbed data can significantly enhance the verifiability of research results and facilitate further research studies in the same area.

We should note that the PPT technique described in this paper is applicable when data is not censored and the predictor and outcome variables are numeric. The data sets with censored, categorical, and ordinal variables will perhaps require different techniques to preserve both privacy and predictive patterns at the same time.

[2]The program generating K-D trees was implemented by Steven Michael of MIT Lincoln Laboratory. We archived the URL of the web site posting this program. See http://www.webcitation.org/5fryepiDd

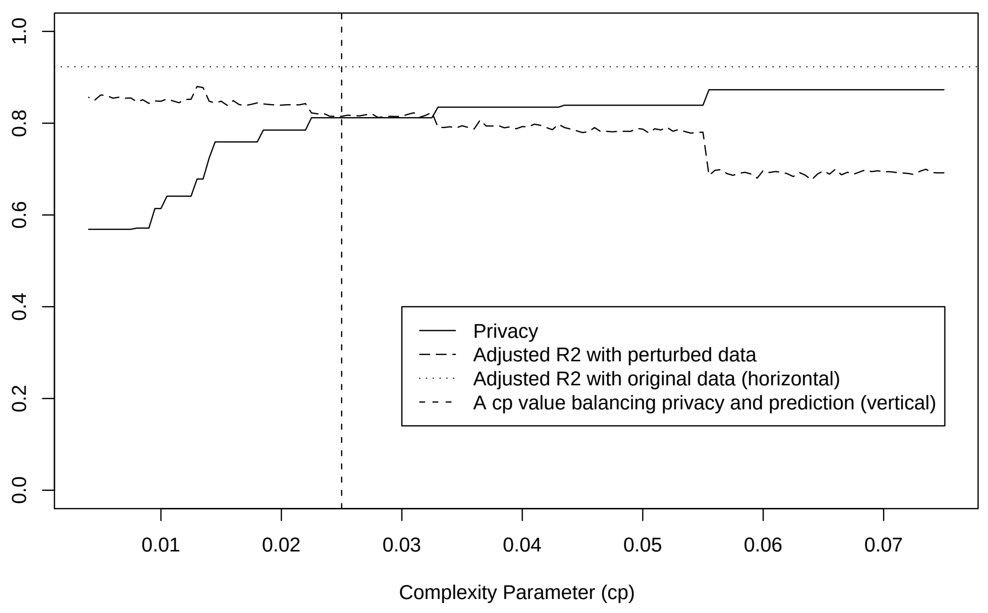

Figure 6: Tradeoff between privacy and prediction performance. Complexity parameter (*cp*) is a parameter used when building trees. Splits that do not decrease the overall lack of fit by *cp* are not attempted. Therefore, typically, smaller *cp*'s provide larger trees with more nodes and levels and, vice versa.

6. CONCLUSION

Benefiting from privacy preserving data perturbation techniques can enable data sharing for proprietary and confidential data, which is commonly found in software engineering research. Research in this direction holds strong potentials to facilitate software engineering research studies in the future.

In this paper, we presented a novel tree-based data perturbation technique that we call Pattern-Preserving Tree-Based Algorithm, or PPT for short. This approach preserves privacy effectively while also preserving the predictive patterns in the original data. As a result, data perturbation using PPT can significantly enhance our capacity to verify the published research results. In addition, it facilitates future research studies investigating similar or related research questions.

The results obtained in this study encourage us to conduct further research on this topic. We plan to work on other data sets to understand the feasibility and effectiveness of our approach and to further generalize and refine it.

7. REFERENCES

[1] R. Agrawal and R. Srikant. Privacy preserving data mining. In *2000 ACM SIGMOD*, pages 439–450, Dallas, TX, May 2000.

[2] R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, pages 217–228, 2005.

[3] B. W. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby. Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering*, 1:57–94, 1995.

[4] L. Breiman, J. H. Freidman, R. A. Olshen, and C. J. Stone. *Classification and regression trees.* Wadsworth, 1984.

[5] G. Canfora and C. Visaggio. Tuning anonymity level for assuring high data quality: an empirical study. *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 91–98, Sept. 2007.

[6] G. Canfora and C. A. Visaggio. Does enforcing anonymity mean decreasing data usefulness? In *QoP '08: Proceedings of the 4th ACM workshop on Quality of protection*, pages 15–22, New York, NY, USA, 2008. ACM.

[7] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, In Press, Corrected Proof:–, 2008.

[8] B. Cukic and Y. Ma. Predicting fault-proneness: Do we finally know how? In *Reliability Analysis of System Failure Data - RAF07*, Cambridge, UK, March 2007.

[9] N. E. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Trans. on Software Engineering*, 26(8):797–814, 2000.

[10] B. C. M. Fung, K. Wang, and P. S. Yu. Top-down specialization for information and privacy preservation. In *ICDE*, pages 205–216, 2005.

[11] R. Gentleman. Reproducible research: a bioinformatics case study. *Statistical Applications in Genetics and Molecular Biology*, 4:Article2, 2005. PMID: 16646837.

[12] F. E. Harrell. Design: Design package, 2005. R package version 2.0-12.

[13] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *ICDM*, pages 99–106, 2003.

[14] T. M. Khoshgoftaar, E. B. Allen, and J. Deng. Using regression trees to classify fault-prone software modules. *IEEE Trans. on Reliability*, 51(4):455–462, 2002.

[15] T. M. Khoshgoftaar, E. B. Allen, W. Jones, and J. P. Hudepohl. Using classification trees for software quality models: Lessons learned. *International Journal of Software Engineering and Knowledge Engineering*, 9(2):217–232, 1998.

[16] A. G. Koru and J. Tian. Comparing high change modules and modules with the highest measurement values in two large-scale open-source products. *IEEE Transactions on Software Engineering*, 31(8):625–642, 2005.

[17] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *SIGMOD*, 2005.

[18] K. Lefevre, D. J. Dewitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *ICDE*, 2006.

[19] X.-B. Li and S. Sarkar. A tree-based data perturbation approach for privacy-preserving data mining. *IEEE Trans. on Knowl. and Data Eng.*, 18(9):1278–1283, 2006.

[20] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. *l*-diversity: Privacy beyond k-anonymity. In *22nd IEEE International Conference on Data Engineering (ICDE 2006)*, Atlanta, Georgia, April 2006.

[21] Promise. Promise data repository, 2007.

[22] R. W. Selby and A. A. Porter. Learning from examples: Generation and evaluation of decision trees for software resource analysis. *IEEE Trans. on Software Engineering*, 14(12):1743–1757, 1988.

[23] S. Spiekermann and L. F. Cranor. Engineering privacy. *IEEE Transactions on Software Engineering*, 35(1):67–82, 2009.

[24] L. Sweeney. K-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[25] R. D. C. Team. R: A language and environment for statistical computing, 2003. ISBN 3-900051-00-3.

[26] T. M. Therneau and B. A. R. port by Brian Ripley. *rpart: Recursive Partitioning*, 2008. R package version 3.1-41.

[27] J. Tian and J. Henshaw. Tree-based defect analysis in testing. *Proc. 4th Int. Conf. on Software Quality*, 1994.

[28] J. Tian, J. Henshaw, and I. Burrows. Analysis of factors affecting in-field quality using tree-based predictive modeling. *Proc. IBM Software Development Conf.*, 1994.

[29] J. Tian and J. Palma. Analyzing and improving reliability: A tree based approach. *IEEE Software*, 15(2):97–104, 1998.

[30] V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *ACM SIGMOD Record*, 33(1):50–57, March 2004.

APPENDIX

Table 1 is the log-transformed original COCOMO data (called COC81 in the PROMISE repository). Table 2 is the PPT-perturbed output using Table 1 and Figure 3-(a) as inputs for D and T in the PPT algorithm shown in Figure 4, respectively.

	rely	data	cplx	time	stor	virt	turn	acap	aexp	pcap	vexp	lexp	modp	tool	sced	loc	actual
1	0.63	0.77	0.53	0.69	0.72	0.77	0.73	0.78	0.76	0.77	0.74	0.69	0.81	0.74	0.71	4.74	7.62
2	0.63	0.77	0.62	0.69	0.72	0.69	0.73	0.69	0.65	0.69	0.64	0.67	0.74	0.69	0.69	5.68	7.38
3	0.69	0.77	0.62	0.69	0.69	0.63	0.66	0.62	0.60	0.62	0.64	0.67	0.65	0.65	0.69	4.89	5.50
4	0.56	0.77	0.53	0.69	0.69	0.63	0.69	0.78	0.65	0.88	0.69	0.67	0.81	0.69	0.71	4.11	5.48
5	0.63	0.66	0.69	0.69	0.69	0.63	0.69	0.69	0.69	0.62	0.64	0.67	0.81	0.69	0.69	2.83	3.53
6	0.56	0.69	0.62	0.69	0.79	0.69	0.69	0.90	0.69	0.88	0.64	0.67	0.81	0.74	0.69	1.61	3.78
7	0.56	0.69	0.69	0.69	0.69	0.63	0.63	0.69	0.69	0.69	0.64	0.67	0.65	0.65	0.69	2.07	2.20
8	0.77	0.66	0.83	0.98	0.94	0.83	0.69	0.54	0.65	0.69	0.79	0.76	0.74	0.74	0.73	3.14	6.98
9	0.77	0.66	0.83	0.83	0.79	0.77	0.69	0.62	0.69	0.62	0.74	0.73	0.65	0.69	0.69	3.43	6.05
10	0.88	0.66	0.83	0.75	0.94	0.69	0.73	0.62	0.60	0.62	0.64	0.69	0.69	0.69	0.69	3.40	5.77
11	0.88	0.66	0.83	0.75	0.94	0.69	0.73	0.62	0.60	0.62	0.64	0.69	0.69	0.69	0.69	3.50	5.39
12	0.77	0.66	0.83	0.75	0.72	0.69	0.69	0.62	0.60	0.62	0.69	0.67	0.65	0.69	0.73	3.64	5.31
13	0.77	0.66	0.83	0.75	0.72	0.77	0.69	0.54	0.69	0.53	0.74	0.69	0.60	0.69	0.69	3.26	4.38
14	0.77	0.66	0.97	0.83	0.94	0.77	0.69	0.62	0.69	0.53	0.74	0.73	0.74	0.81	0.80	1.39	4.30
15	0.88	0.66	0.83	0.83	0.72	0.77	0.63	0.62	0.76	0.62	0.79	0.76	0.65	0.69	0.80	1.59	4.13
16	0.88	0.69	0.83	0.83	0.94	0.69	0.63	0.62	0.69	0.62	0.69	0.69	0.69	0.69	0.69	1.96	3.71
17	0.88	0.69	0.83	0.83	0.94	0.69	0.63	0.62	0.60	0.62	0.69	0.69	0.69	0.69	0.69	1.53	2.30
18	0.77	0.77	0.77	0.83	0.79	0.69	0.73	0.62	0.69	0.69	0.69	0.69	0.81	0.74	0.73	5.77	9.34
19	0.77	0.73	0.69	0.75	0.79	0.63	0.66	0.54	0.65	0.69	0.69	0.69	0.65	0.65	0.69	7.05	8.79
20	0.88	0.73	0.83	0.75	0.79	0.77	0.73	0.54	0.60	0.73	0.74	0.73	0.81	0.69	0.73	5.70	8.76
21	0.69	0.77	0.77	0.72	0.76	0.63	0.63	0.62	0.69	0.69	0.69	0.69	0.65	0.65	0.69	5.53	7.81
22	0.77	0.69	0.69	0.82	0.72	0.69	0.69	0.62	0.60	0.62	0.64	0.69	0.65	0.69	0.80	4.78	6.59
23	0.77	0.69	0.69	0.73	0.72	0.69	0.69	0.62	0.60	0.62	0.64	0.69	0.69	0.69	0.80	4.36	6.29
24	0.63	0.69	0.62	0.72	0.72	0.69	0.63	0.69	0.83	0.69	0.74	0.67	0.60	0.60	0.69	4.51	6.12
25	0.77	0.77	0.83	0.77	0.72	0.69	0.63	0.62	0.69	0.62	0.74	0.69	0.60	0.65	0.73	3.66	6.26
26	0.66	0.69	0.62	0.73	0.72	0.77	0.73	0.62	0.69	0.62	0.74	0.69	0.65	0.74	0.73	3.89	5.96
27	0.77	0.66	0.77	0.85	0.79	0.69	0.63	0.69	0.69	0.69	0.69	0.69	0.60	0.74	0.73	2.34	4.49
28	0.77	0.73	0.83	0.75	0.79	0.77	0.73	0.62	0.69	0.62	0.74	0.73	0.74	0.74	0.69	2.64	4.60
29	0.63	0.69	0.69	0.69	0.69	0.69	0.69	0.74	0.83	0.62	0.69	0.69	0.65	0.65	0.80	1.14	2.12
30	0.63	0.69	0.69	0.69	0.69	0.69	0.69	0.69	0.83	0.62	0.69	0.69	0.65	0.65	0.80	1.09	1.93
31	0.88	0.73	0.69	0.91	0.94	0.77	0.73	0.62	0.60	0.62	0.74	0.73	0.69	0.69	0.69	4.14	6.97
32	0.63	0.73	0.62	0.69	0.69	0.69	0.69	0.54	0.60	0.69	0.69	0.69	0.74	0.74	0.69	5.97	6.56
33	0.88	0.73	0.83	0.91	0.94	0.77	0.66	0.62	0.60	0.62	0.64	0.69	0.65	0.65	0.69	3.76	6.41
34	0.77	0.73	0.69	0.72	0.69	0.69	0.63	0.69	0.69	0.69	0.69	0.69	0.65	0.74	0.80	3.18	5.44
35	0.56	0.66	0.83	0.72	0.79	0.77	0.69	0.69	0.65	0.69	0.74	0.69	0.81	0.81	0.69	2.64	4.42
36	0.63	0.73	0.62	0.69	0.69	0.63	0.63	0.78	0.69	0.77	0.64	0.67	0.69	0.65	0.71	2.77	4.03
37	0.63	0.66	0.53	0.69	0.72	0.69	0.69	0.62	0.60	0.62	0.69	0.69	0.69	0.69	0.69	4.11	3.87
38	0.69	0.69	0.77	0.69	0.69	0.63	0.63	0.54	0.65	0.69	0.64	0.67	0.60	0.65	0.69	2.77	2.56
39	0.69	0.69	0.77	0.69	0.69	0.63	0.69	0.54	0.60	0.53	0.69	0.67	0.65	0.74	0.69	1.97	2.20
40	0.69	0.66	0.83	0.69	0.69	0.69	0.63	0.62	0.60	0.77	0.69	0.69	0.74	0.69	0.69	1.39	2.20
41	0.63	0.66	0.69	0.69	0.69	0.63	0.63	0.69	0.60	0.53	0.64	0.67	0.65	0.65	0.69	1.84	1.95
42	0.63	0.71	0.73	0.69	0.72	0.63	0.73	0.62	0.69	0.66	0.64	0.67	0.67	0.67	0.71	3.84	3.83
43	0.69	0.71	0.73	0.69	0.79	0.63	0.73	0.62	0.69	0.69	0.64	0.67	0.69	0.69	0.71	3.39	4.43
44	0.63	0.71	0.73	0.72	0.79	0.63	0.73	0.69	0.69	0.69	0.64	0.67	0.74	0.69	0.71	3.45	4.48
45	0.63	0.71	0.73	0.69	0.72	0.63	0.73	0.69	0.69	0.69	0.64	0.67	0.69	0.67	0.71	3.58	4.67
46	0.63	0.71	0.73	0.69	0.72	0.63	0.73	0.69	0.69	0.62	0.64	0.67	0.69	0.69	0.71	4.30	4.84
47	0.56	0.66	0.83	0.69	0.69	0.63	0.63	0.54	0.60	0.53	0.74	0.73	0.74	0.69	0.71	3.18	3.61
48	0.63	0.66	0.62	0.69	0.69	0.63	0.69	0.78	0.65	0.77	0.64	0.67	0.74	0.69	0.71	6.14	7.15
49	0.69	0.69	0.62	0.69	0.69	0.69	0.63	0.54	0.69	0.53	0.74	0.69	0.60	0.65	0.69	4.52	5.06
50	0.77	0.69	0.69	0.83	0.79	0.69	0.63	0.62	0.69	0.62	0.74	0.69	0.69	0.69	0.69	3.22	5.18
51	0.63	0.69	0.69	0.69	0.69	0.69	0.77	0.78	0.69	0.88	0.69	0.67	0.81	0.74	0.71	2.40	4.81
52	0.63	0.66	0.62	0.69	0.72	0.77	0.69	0.69	0.69	0.69	0.74	0.73	0.81	0.74	0.69	2.22	3.74
53	0.63	0.66	0.77	0.75	0.79	0.83	0.69	0.54	0.69	0.53	0.74	0.73	0.69	0.74	0.73	1.84	2.71
54	0.69	0.66	0.69	0.69	0.72	0.77	0.63	0.69	0.60	0.69	0.69	0.67	0.65	0.74	0.69	1.69	3.04
55	0.63	0.66	0.53	0.69	0.69	0.63	0.63	0.62	0.60	0.77	0.64	0.67	0.74	0.69	0.69	1.99	2.94
56	0.77	0.66	0.83	0.83	0.79	0.69	0.69	0.62	0.65	0.69	0.74	0.73	0.74	0.74	0.73	3.33	6.87
57	0.69	0.66	0.77	0.75	0.79	0.83	0.69	0.69	0.69	0.69	0.74	0.73	0.74	0.74	0.80	2.89	5.47
58	0.88	0.66	0.83	0.98	0.79	0.69	0.69	0.54	0.60	0.53	0.64	0.67	0.65	0.69	0.69	3.26	4.88
59	0.69	0.66	0.77	0.72	0.72	0.69	0.63	0.69	0.69	0.69	0.69	0.69	0.65	0.69	0.69	3.18	4.26
60	0.77	0.66	0.83	0.75	0.72	0.69	0.69	0.62	0.76	0.62	0.74	0.73	0.74	0.74	0.73	2.04	4.06
61	0.69	0.66	0.77	0.69	0.69	0.63	0.63	0.62	0.69	0.62	0.64	0.69	0.60	0.69	0.69	3.37	3.93
62	0.63	0.66	0.83	0.75	0.79	0.77	0.69	0.58	0.60	0.53	0.79	0.76	0.65	0.81	0.69	2.31	3.66
63	0.69	0.66	0.77	0.69	0.69	0.69	0.63	0.54	0.60	0.62	0.69	0.69	0.60	0.69	0.69	2.40	2.77

Table 1: Original COCOMO 81 Data Set (Log Transformed and Rounded to Two Fractional Digits)

	rely	data	cplx	time	stor	virt	turn	acap	aexp	pcap	vexp	lexp	modp	tool	sced	loc	actual
1	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.76	0.82
2	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.76	0.79
3	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.76	0.59
4	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.59
5	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.40	0.38
6	0.81	0.9	0.75	0.71	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.23	0.41
7	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.29	0.24
8	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.51	0.75
9	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.65
10	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.62
11	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.58
12	0.81	0.9	0.75	0.82	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.57
13	0.81	0.9	0.75	0.82	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.47
14	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.20	0.46
15	0.81	0.9	0.75	0.82	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.23	0.44
16	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.28	0.40
17	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.22	0.25
18	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.76	1.00
19	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.94	0.94
20	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.76	0.94
21	0.81	0.9	0.75	0.71	0.76	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.76	0.84
22	0.81	0.9	0.75	0.82	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.76	0.71
23	0.81	0.9	0.75	0.82	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.67
24	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.51	0.65
25	0.81	0.9	0.75	0.82	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.51	0.67
26	0.81	0.9	0.75	0.77	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.51	0.64
27	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.33	0.48
28	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.37	0.49
29	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.16	0.23
30	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.15	0.21
31	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.75
32	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.76	0.70
33	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.51	0.69
34	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.51	0.58
35	0.81	0.9	0.75	0.71	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.37	0.47
36	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.39	0.43
37	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.41
38	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.39	0.27
39	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.28	0.24
40	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.20	0.24
41	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.26	0.21
42	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.51	0.41
43	0.81	0.9	0.75	0.71	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.47
44	0.81	0.9	0.75	0.71	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.48
45	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.51	0.50
46	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.52
47	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.39
48	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.94	0.77
49	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.84	0.89	0.55	0.54
50	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.55
51	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.34	0.52
52	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.31	0.40
53	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.26	0.29
54	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.24	0.33
55	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.28	0.32
56	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.51	0.73
57	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.41	0.59
58	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.52
59	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.46
60	0.81	0.9	0.75	0.82	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.29	0.43
61	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.51	0.42
62	0.81	0.9	0.75	0.82	0.90	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.93	0.89	0.33	0.39
63	0.81	0.9	0.75	0.71	0.75	0.84	0.89	0.71	0.8	0.74	0.88	0.91	0.86	0.85	0.89	0.34	0.30

Table 2: Perturbed COCOMO 81 Data Set (Rounded to Two Fractional Digits)