2014

LACE: Supporting Privacy-Preserving Data Sharing in Transfer Defect Learning

Fayola Peters

LACE: Supporting Privacy-Preserving Data Sharing in Transfer Defect Learning

Fayola Peters

Dissertation submitted
to the College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in
Computer Science

Tim Menzies, Ph.D., Chair
Arun Ross, Ph.D.
Katerina Goseva-Popstojanova, Ph.D.
Bojan Cukic, Ph.D.
Mark Grechanik, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2014
Keywords: Privacy Preserving Data Sharing, Defect Prediction, Software Engineering, Clojure

Abstract

LACE: Supporting Privacy-Preserving Data Sharing in Transfer Defect Learning

Cross Project Defect Prediction (CPDP) is a field of study where an organization lacking enough local data can use data from other organizations or projects for building defect predictors. Research in CPDP has shown challenges in using "other" data, therefore transfer defect learning has emerged to improve on the quality of CPDP results. With this new found success in CPDP, it is now increasingly important to focus on the privacy concerns of data owners.

To support CPDP, data must be shared. There are many privacy threats that inhibit data sharing. We focus on sensitive attribute disclosure threats or attacks, where an attacker seeks to associate a record(s) in a data set to its sensitive information. Solutions to this sharing problem comes from the field of Privacy Preserving Data Publishing (PPDP) which has emerged as a means to confuse the efforts of sensitive attribute disclosure attacks and therefore reduce privacy concerns. PPDP covers methods and tools used to disguise raw data for publishing. However, prior work warned that *increasing* data privacy *decreases* the efficacy of data mining on privatized data.

The goal of this research is to help encourage organizations and individuals to share their data publicly and/or with each other for research purposes and/or improving the quality of their software product through defect prediction. The contributions of this work allow three benefits for data owners willing to share privatized data: 1) that they are fully aware of the sensitive attribute disclosure risks involved so they can make an informed decision about what to share, 2) they are provided with the ability to privatize their data and have it remain useful, and 3) the ability to work with others to share their data based on what they learn from each others data. We call this private multiparty data sharing.

To achieve these benefits, this dissertation presents LACE (Large-scale Assurance of Confidentiality Environment). LACE incorporates a privacy metric called IPR (Increased Privacy Ratio) which calculates the risk of sensitive attribute disclosure of data through comparing results of queries (attacks) on the original data and a privatized version of that data. LACE also includes a privacy algorithm which uses intelligent instance selection to prune the data to as low as 10% of the original data (thus offering complete privacy to the other 90%). It then mutates the remaining data making it possible that over 70% of sensitive attribute disclosure attacks are unsuccessful. Finally, LACE can facilitate private multiparty data sharing via a unique leader-follower algorithm (developed for this dissertation). The algorithm allows data owners to serially build a privatized data set, by allowing them to only contribute data that are not already in the private cache. In this scenario, each data owner shares even less of their data, some as low as 2%.

The experiments of this thesis, lead to the following conclusion: at least for the defect data studied here, *data can be minimized, privatized and shared without a significant degradation in utility*. Specifically, in comparative studies with standard privacy models (k-anonymity and data swapping), applied to 10 open-source data sets and 3 proprietary data sets, LACE produces privatized data sets that are significantly smaller than the original data (as low as 2%). As a result LACE offers better protection against sensitive attribute disclosure attacks than other methods.

Acknowledgments

For all his help with this thesis, I am grateful to Dr. Tim Menzies. As my advisor, he has shown a tremendous faith in my abilities and shared not only his expert knowledge in the field of Data Mining, but also advised me on practical issues concerning my future career plans and what I would need to do now and in the near future to be competitive.

Much of the research in this thesis was conducted at the West Virginia University Modelling Intelligence Lab (the MIL).

Finally, I would like to thank my family and friends. My parents and my brothers and sisters who have done all that they can to support me on this path, and my friends who have kept me laughing, entertained and fed throughout this entire process.

Contents

List of Figures

List of Tables

List of Program Codes

Chapter 1

Introduction

Previously, research into software engineering was hampered by a lack of data. That era is over. Recent advances in the mining of software repositories has created tremendous opportunities to identify interesting trends and patterns about the process of software development. For example, in the field of Cross Project Defect Prediction (CPDP), researchers have found it possible to predict defects for software projects with insufficient data by using data from other projects [16–29]. Considering that inadequate software testing costs the US economy $59.5 billion per year, even though 50% to 80% of development budgets go toward testing [30], this result in CPDP can contribute to improved software inspection efficiency [31] and improved software quality. However although the field of CPDP is useful and active, it's main component is *data sharing* which brings up privacy concerns.

Very few studies in CPDP focus on privacy [11, 12]. Instead, more attention is paid to other issues such as: finding or creating viable data for building quality defect predictors and doing so with little computational expense. In the future as the number and size of software products increase and we seek to find out what we can learn from each other, addressing privacy will become even more important. This is evident by the release of the *privatized Google data set of testing results*. These contain 3.5 Million test suite execution results. When questioned about sharing the

source code being tested and details on the failures, data owners responded with:

> *Sharing industrial datasets with the research community is extremely valuable, but also extremely challenging as it needs to balance the usefulness of the dataset with the industry's concerns for privacy and competition [32].*

Research has successfully expanded to show that cross project defect prediction is now possible between open source and proprietary projects [28]. Hence, if we can address privacy concerns, there is much the open source community and proprietary developers can learn from each other.

As cross project research moves in this new direction it is now extremely vital for new studies to focus on maintaining the confidentiality[1] [11, 12] of data with privacy-preserving methods. Failure in this regard would mean that even with the successes of cross project defect prediction [18, 20, 24, 28], this promising field of research can be stalled.

Privacy-Preserving Data Publishing (PPDP) also referred to as *data sharing* in this dissertation, has emerged as a means to minimize the problems that can be caused by attackers (those seeking to gain confidential knowledge from published data) and therefore reduce privacy concerns. PPDP covers methods and tools used to disguise raw data for publishing.

Research in PPDP has two key goals: 1) to publish data that are private such that little or no knowledge is gained about the original data, and 2) useful for tasks such as classification or aggregate querying answering. As a result of these goals, there are many components to privacy research. These are: privacy threats, privacy models, privacy techniques, privacy algorithms, privacy measures and utility (Chapter 2 explains the details of these components). First researchers determined the possible privacy threat(s), then decide on a definition for privacy (model), this in turn determines the techniques to use and algorithms are then created. In some cases the model also determines how privacy is measured. Finally utility is a measure of the usefulness of the shared data.

[1] The term *confidentiality* in this dissertation refers to maintaining privacy of data by limiting access to data through methods that minimize and obfuscate data making it distinguishable from the original.

Many of the privacy tools focus on protecting the sensitive personal data of an individual, such as social security numbers or a health diagnosis. While the data we study in this dissertation are code metrics extracted from source code at the granularity level of classes. All the values are numeric except for the class label. A successful attack on this data can expose the complexity of the source code for a particular class which can lead the attacker to infer the effort and cost required to maintain said code. Such a breach can have negative effects particularly in cases where software engineering companies are engaged in competitive bidding for contracts.

Sensitive attribute disclosure is the privacy threat we focus on in this dissertation. To carry out this threat *background knowledge* about a specific record in the data are required. With this knowledge the record can be found as well as the sensitive information associated with that record. Standard methods for privacy-preserving data sharing like k-anonymity [33, 34] do not protect against any background knowledge used to gain information from privatized data, and so may still reveal the sensitive attribute of a record. Moreover, two reports concluded that the *more* we privatize data, the *less* useful it becomes for the utility of certain tasks, for example, *classification*. Grechanik et al. and Brickell et al. [3,35] reported that the application of standard privacy methods such as k-anonymity [33,34], l-diversity [36], and t-closeness [37] damages inference power as the privacy of the data increases.

The above motivates a need for privacy-preserving data sharing solutions for cross project defect prediction to encourage data sharing for research in detecting useful trends and patterns in software engineering and the improvement of inspection efficiency and software quality. This dissertation presents one such solution.

1.1 LACE: Supporting Privacy-Preserving Data Sharing in Transfer Defect Learning

This dissertation presents and evaluates LACE, a tool that facilitates privacy-preserving data sharing for two scenarios:

1. The case where one data owner wants to share their data, and;
2. The case where multiple data owners want to collaborate with each other and share their data collectively.

LACE combines data minimization and constrained obfuscation algorithms to produce privatized data candidates that are both private and useful for data sharing (publication). The main goal of LACE is to prevent or lower the risk of a sensitive attribute disclosure attack (explained in Section 2.1). As a result, LACE also includes a means to measure the level of sensitive attribute disclosure of a privatized data candidate. This is called IPR, the **I**ncreased **P**rivacy **R**atio. We cannot claim absolute data privacy with LACE, however, with IPR we can show a data owner how protected the sensitive attribute values of their data will be against a sensitive attribute disclosure attack. It is then up to them to decide whether or not to share their data based on the IPR.

Our approach to privacy-preserving data sharing relies on two key insights to achieve a balance between privacy and utility. First, **only a subset of the data is required to get comparable utility with the original data**. Research in minimization techniques yielded algorithms that avoid the drawbacks of large data set analysis such as large storage requirements and high computational expense. This is accomplished by selecting exemplars (instances that best describe the data) from the data and using only these exemplars in analysis. An additional benefit is that in the context of privacy, since only the exemplars are used for analysis, data that are not exemplars are kept private by the data owner. We apply a minimization algorithm called *CLIFF* (Section 4.2.1) which determines these exemplars by ranking the values based on how well they predict for a class value

in the data and selecting only those instances that have the highest ranks.

The second insight is: **Providing that noise does not drive data across classification boundaries, adding noise to the independent variables of a minimized data set has minimal effect on classification**. As evidence for this claim, in the experiments of this dissertation, we explore "constrained obfuscation"; i.e. we find the hyperspace classification boundary and mutate by a random amount up to, but not more than, the distance to that boundary, then we can successfully obfuscate the data without damaging data mining efficacy. This insight is crucial in creating a privacy algorithm for defect data that respects class boundaries. Therefore we propose *MORPH* (Section 4.2.2), an algorithm that takes advantage of this insight and randomly changes the data values up to the point of class boundaries where class values change.

1.2 Thesis Statement

Preliminary work [11, 12] has shown promise with CLIFF&MORPH. These algorithms created privatized data candidates that are both private (protects against sensitive attribute disclosure attacks) and useful for cross project defect prediction. With these insights and results in mind, we make the following claim:

> **Thesis Statement:** Privacy-Preserving Data Sharing in Transfer Defect Learning can be accomplished by the Minimization and then Constrained Obfuscation of data.

1.3 Contributions and Outline

The main contributions of this dissertation are:

1. **Minimizing and Obfuscating Data:** A tool which we call LACE (Large-scale Assurance of Confidentiality Environment) which consists of two algorithms for privacy: CLIFF and MORPH. We use CLIFF to find exemplars and therefore mitigate against the drawbacks

of poor data quality and MORPH changes the values of exemplars found by CLIFF while keeping them in behind class boundaries (Section 4.2).

2. **Increased Privacy Ratio (IPR):** LACE also includes IPR, a means to measure the level of protection offered against sensitive attribute disclosure attacks (Section 4.3).

3. **Locating Optimized Privatized Data Candidates:** A means to obtain improved privacy algorithms via parameter tuning thereby removing the burden of parameter value decisions from the data owner (Chapter 5).

4. **Private Multiparty Data Sharing:** LACE facilitates multiparty data sharing where multiple data owners can collaborate to create a collective privatized data candidate. In this scenario each data owner uses output (shared data) from another in order to privatize their own data (Chapter 6).

5. **Proprietary Predicts for Open Source:** Empirical evidence that it is possible for proprietary project defect data to build defect predictors for open source projects.

6. **Better CPDP:** Later work in this dissertation solves a problem of high false alarm rates found in an earlier publication from this work (Section 6.3.2).

7. **Functional Programming for Privacy:** A lesser contribution of this work is that all code used to build LACE are done in a functional language called Clojure [38]. Clojure targets the Java Virtual Machine and provides easy access to the Java frameworks. As a result of this, LACE was faster to code in Clojure than Java. LACE can easily be extended with either Java or Clojure libraries.

We begin with background and related work for privacy preserving data publishing and defect prediction in Chapter 2 and Chapter 3 respectively. Next, we explain the design and operation of LACE in Chapter 4, followed by experiments in Chapter 5 and Chapter 6. Finally, we end this

dissertation with threats to validity in Chapter 7 and conclusions in Chapter 8. We have a total of five research questions (RQ1 to RQ5) and will briefly describe our approaches to answering each in the following sections. We then conclude this chapter with publications that support this dissertation.

1.4 Minimizing and Obfuscating Data

In the literature, minimizing data has many synonyms; data summary, instance-based selection, exemplar-based learning, relevancy filtering, prototype learning and so on. It is seen as a means to solve some of the drawbacks of analyzing large data sets, such as high storage and computation requirements. In addition, if data is noisy, this can have an effect on inference [39, 40] and minimizing data has proven to be an effective immunization solution to noise. In Section 4.2.1 we present a minimization technique called CLIFF which avoids the drawbacks for analyzing data and also seeks those data points that best predict for a target class in order to maintain comparable utility with the original data. For clarity, we refer to data resulting from a minimization technique as *exemplars*.

In addition to minimizing data, we seek to further protect the data by obfuscating the exemplars. There are a plethora of privacy algorithms available to accomplish this task, however many share the problem of reducing the usefulness of data as privacy increases [3, 35]. In Section 4.2.2 we present MOPRH, a privacy algorithm that is designed to maintain the structure of the data in order to maintain the usefulness of the data. In combination, CLIFF&MORPH represent the privacy algorithm in LACE. We benchmark its performance with other state-of-the-art privacy algorithms in Chapter 5 and answer the following research question.

RQ1: Does CLIFF&MORPH provide better balance between privacy and utility than other state-of-the-art privacy algorithms?

CLIFF&MORPH like other privacy algorithms relies on a set of "magic parameters" that con-

trol certain engineering decisions within that system. Different parameter values can lead to different privatized data candidates with varying results for privacy and utility. Therefore, to better answer this question (RQ1) we perform a parameter tuning experiment with different parameter values for the privacy algorithms studied here. This approach extends on previous work [11, 12] which uses a narrow range of parameter values. This experimental approach raises the next two research questions.

RQ2: How hard is parameter tuning for privacy algorithms?

With multiple privacy algorithms and even more possible parameter vectors, our tunings require multiple runs of the systems to assess the impact of a particular range of the *magic* parameters. We found that in the privacy domain, a large number of tunings or search budget is unnecessary. We were able to find good privatized data candidates in as little as eight runs for each privacy algorithm studied.

RQ3: Are the results for parameter tuning for privacy algorithms useful for reducing the search budget (multiple system runs)?

Here we investigate if the parameter vectors found from answering RQ2 can work well for other problems and reduce the search budget? Since the data owners' search ends when they are satisfied with a particular result, the ability to transfer parameter knowledge would reduce any search budget. We found that we could successfully transfer parameters learned from one data set to another.

1.5 Private Multiparty Data Sharing

Beyond data privacy for a single data set, LACE extends to facilitate data privacy for multiple data owners based on the following scenario. Consider the problem of l parties (data owners) $P_0 ... P_{l-1}$, each with local data, x_i. They want to securely work together to create a *private cache* containing pooled minimized and obfuscated data from all parties involved. Each data owner P_i determines

what data to add to the private cache based on what others have added previously. The final private cache can then be shared for cross project defect prediction. According to Lindell et al. [41], this problem is a special case in cryptography where a set of parties with private data wishes to jointly compute some function of their data. This joint computation should have the property that the parties learn the correct output and nothing else, even if some of the parties maliciously collude to obtain more information.

Given that our aim is to share data for the purpose of cross project defect prediction, we incorporate a unique leader-follower algorithm (developed for this dissertation) with LACE which determines what sub-set of data each data owner can add to the private cache.

In Chapter 6, we evaluate private multiparty data sharing with LACE using two research questions:

RQ4: Does private multiparty data sharing with LACE offer protection against sensitive attribute disclosure for each data owner?

Prior to submitting exemplars to the private cache, each data owner calculates the *increased privacy ratio* (IPR) [12] of the exemplars to determine how much of the sensitive information in the original data are revealed by the exemplars. IPR is discussed in Section 4.3.

RQ5: Are the data resulting from private multiparty data sharing i.e. the private cache, useful for cross project defect prediction?

Recent results in cross project defect prediction have improved due to transfer learning techniques [18, 24, 27, 28]. We therefore measure *usefulness* by finding out if there is a significant difference in the performance of defect predictors built with all the original data from the data owner vs. LACEd data. For each test set used, we apply the transfer learning technique of Turhan et al. [18] by finding a sub-set of the private cache that are most similar to the test set.

1.6 Publications Supporting Thesis

Table 1.1 shows that the primary findings of our work thus far that have been published at major software engineering research venues or are currently under review. Some of the content appear in this dissertation.

Table 1.1: Publications supporting this research.

Contributions	Venue	Year	Title
1	ICSE	2012	Privacy and Utility for Defect Prediction: Experiments with "MORPH" [11]
1	TSE	2013	Balancing Privacy and Utility in Cross-Company Defect Prediction [12]
1	ESEM	2013	Learning from Open-Source Projects: An Empirical Study on Defect Prediction [28]
2	ASE Journal	under review	Parameter Tuning for Balancing Privacy and Utility in Cross Defect Prediction

Chapter 2

Privacy Preserving Data Publishing

Our objective for this work is to encourage data sharing. We seek to give data owners the means to maintain the confidentiality stipulations of their data by producing *privatized data candidate(s)* for publication. These privatized data candidates are disguised versions of the original data. Ideally these disguised data are *useful* for research purposes and ***do not*** *reveal any information about the original data*.

In this chapter we elaborate on the major components that are involved in data privacy research, namely: 1) privacy threats, 2) privacy models (definitions), 3) privacy techniques, 4) privacy algorithms and finally 5) the utility of the privatized data candidates. In addition, we explain how to evaluate privacy in Section 2.5 and we also look at data privacy research in another area of software engineering, i.e. testing and debugging software artifacts (Section 2.6). Finally we deal with utility in Chapter 3 where we focus on the utility measure used in this dissertation, i.e. cross project defect prediction with transfer learning.

Before elaborating on these five major components, it is important to know how data is considered in privacy-preserving data publishing. More specifically, we focus on the software defect data studied in this work. Defect data consists of a set of classes which we refer to as targets (T=$\{t_1, t_2, ..., t_{|}T|\}$). Each target $t \in$ T is a tuple of attribute values representing the individual

target class. Each attribute in the set of attributes (A) could have one or more of the following definitions:

- *Direct-identifiers* are attributes that explicitly identifies an individual or project such as a social security number or filename.
- *Quasi-identifiers (QIDs)* are attributes, $QID \in A$, that in combination, can be used to re-identify an individual target in a data set.
- *Sensitive Attributes (S)* are attributes, $S \in A$, that we do not want an attacker (adversary) to associate with a target, t in a data set.
- *Dependent Attributes* are attributes used when evaluating the utility of data via classification. In this work, utility is measured via *cross project defect prediction*. In other words for targets with unknown dependent attribute values, we predict those values.

Privacy threats represent the different attacks data can face. Threats are based on the top three attributes described above and are classified as *identity disclosure or re-identification*, *membership disclosure*, and *attribute disclosure or sensitive attribute disclosure* [35, 42, 43]. We expand on these in the following section.

2.1 Privacy Threats

Most of the research done in data privacy focuses on protecting micro data, that is, the data of a person. In our work we seek to protect project defect data of software projects. Here instead of persons we have *classes* which we also refer to as targets, and the data about each class includes metrics such as WMC (weighted methods per class) or DIT (depth of inheritance tree) which are indicators of the complexity of the code in each target and what effort is require for maintaining the code. For instance with both WMC and DIT, the higher these values are the more complex the code. Details abouts these attributes and others used in this dissertation are shown in Table 2.1.

Table 2.1: Description of the Static Code Metrics Used For Defect Prediction. Jureczko et al. [14, 15] provide more information on these metrics.

Attributes	Symbols	Description
average method complexity	amc	e.g., number of JAVA byte codes
average McCabe	avg_cc	average McCabe's cyclomatic complexity seen in class
afferent couplings	ca	how many other classes use the specific class
cohesion amongst classes	cam	summation of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods
coupling between methods	cbm	total number of new/redefined methods to which all the inherited methods are coupled
coupling between objects	cbo	increased when the methods of one class access services of another
efferent couplings	ce	how many other classes is used by the specific class
data access	dam	ratio of the number of private (protected) attributes to the total number of attributes
depth of inheritance tree	dit	provides the position of the class in the inheritance tree
inheritance coupling	ic	number of parent classes to which a given class is coupled (includes counts of methods and variables inherited)
lack of cohesion in methods	lcom	number of pairs of methods that do not share a reference to an instance variable
another lack of cohesion measure	locm3	if m, a are the number of $methods, attributes$ in a class number and $\mu(a)$ is the number of methods accessing an attribute, then $lcom3 = ((\frac{1}{a}\sum_j^a \mu(a_j)) - m)/(1-m)$
lines of code	loc	measures the volume of code
maximum McCabe	max_cc	maximum McCabe's cyclomatic complexity seen in class
functional abstraction	mfa	number of methods inherited by a class plus number of methods accessible by member methods of the class
aggregation	moa	count of the number of data declarations (class fields) whose types are user defined classes
number of children	noc	measures the number of immediate descendants of the class.
number of public methods	npm	counts all the methods in a class that are declared as public. The metric is known also as Class Interface Size (CIS)
response for a class	rfc	number of methods invoked in response to a message to the object
weighted methods per class	wmc	the number of methods in the class (assuming unity weights for all methods).
number of bugs	bug	number of bugs detected in the class.

When protecting a person's data from privacy threats, the goal is to avoid *re-identification* and block malicious intruders from trying to uncover private data. Re-identification occurs when an intruder with external information such as a voters list, can re-identify an individual from data that has been stripped of personally identifiable information such as a social security number. Popular examples of this are the re-identification of William Weld from released health-care data [33] and Thelma Arnold from the AOL search data [44].

Membership disclosure is another privacy threat that focuses on protecting a person's micro data. It can happen if an attacker is able to confirm that the target's data is contained in a particular data set. For example, if the data set contains information only on HIV patients, then the attacker can infer that the *target* is HIV-positive [43].

In this dissertation, the data are aggregated at the project level and do not contain personnel or specific company information. Hence, re-identification or membership disclosure are not explored further in this work. Instead, we protect the project data for the final privacy threat: *attribute disclosure or sensitive attribute disclosure*. Sensitive attribute disclosure occurs when a target is associated with information about their sensitive attributes, such as software code complexity. The defect data used in this work contains a few attributes that can be considered as *sensitive*, some of these are:

- Weighted Methods Complexity (WMC): WMC is the number of methods in a class (assuming unity weights for all methods). Therefore, larger values of WMC mean large complexity as well.

- Depth of Inheritance Hierarchy (DIT): DIT provides the position of the class in the inheritance tree. This is an indicator of the number of ancestors of a class. Developers and testers may need to understand all ancestors in order to know all the specializations of a class. This is necessary to maintain the software product or to uncover defects.

- Number of Child Classes (NOC): NOC measures the number of immediate descendants of

the class. The number of children represents the number of uses of a class. Therefore, understanding all children is important to understanding the parent. A large number of children increases the burden on developers and testers in understanding, maintaining, and finding defects.

- Response for Class (RFC): The size of the response set for the class includes methods in the class inheritance hierarchy and methods that can be invoked on other objects. The RFC metric counts the number of methods in the response set for a class, which includes the number of local methods and the number of remote methods invoked by local methods. The class that has a large number of responsibilities tends to be large and has many interactions with other classes. Such classes are complex and take more time and effort to maintain and test than smaller classes.

- Coupling between Objects (CBO): the CBO metric counts the number of other classes to which a class is coupled. Larger values of CBO metrics mean that the class is highly coupled. The developers and testers can infer that the maintainability and testability of highly coupled classes is difficult. This makes the process of maintaining and finding defects difficult as well.

- Lines of Code (LOC): Measures the volume of code. It is also seen as a function of code complexity where higher LOC indicate higher complexity. High LOC also make understanding and maintaining code more difficult for developers and testers to find defects.

2.2 Privacy Models

A privacy requirement or goal seeks to avoid privacy breaches or disclosure. In order to realize privacy breaches, one needs to define what constitutes a privacy breach for a particular data set. There are different levels of privacy. Some levels are determined by individuals in the data set or

by the creators of a privacy policy or laws.

An optimal result of a privacy model is defined by Dalenius [45] where he states that access to published data should not enable the attacker to learn anything extra about any target victim, compared to no access to the database, even with the presence of any attacker's background knowledge obtained from other sources [42].

Fung [42], considers two categories of privacy models:

1. A privacy threat occurs when an attacker is able to link a record owner to a sensitive attribute in a published data table. These are specified as, record linkage, attribute linkage and table linkage respectively.

2. The published data should provide the attacker with little additional information beyond their background knowledge.

Brickell [35], defines a privacy model based on sensitive attribute disclosure which occurs when the attacker learns information about an individual's sensitive attributes. In other words, it captures the gain in the attackers' knowledge due to his/her observations of the disguised data set. Also "Microdata privacy can be understood as prevention of membership disclosure" where the attacker should not learn whether a particular individual is included in the database.

In 2007, Wang [46] put forward other definitions of privacy. The article explains:

> *There have been two types of privacy concerning data mining. The first type of privacy, called output privacy, is that the data is minimally altered so that the mining result will not disclose certain privacy. The second type of privacy, called input privacy, is that the data is manipulated so that the mining result is not affected or minimally affected.*

One of the earliest privacy models is *k*-anonymity [33, 34]. This model is widely studied and understood. It requires that for each target in a data set, it's quasi-identifiers must be the same as *k-1* others. For this dissertation, we consider the privacy model based on sensitive attribute

disclosure as described by Brickell et al. [35], where an attacker is unsuccessful at *gaining more* information from a privatized data candidate. Our measure of *gain* is IPR (Section 4.3) based on the privacy measures explained by Brickell et al. [35].

Once a decision is made on the privacy threat to protect their data against, its time to determine what privacy techniques will help accomplish this and create privacy algorithms based on these techniques.

2.3 Privacy Techniques

The idea of disguising a data set is known as anonymization. Since the data used in this dissertation is software defect data and not personal data, we use the term *privatization*. This is performed on the original data set to "satisfy a specified privacy requirement" [42] resulting in a modified data set being published. There are six general categories for anonymization, 1) generalization, 2) suppression, 3) bucketization 4) anatomization, 5) permutation, and 6) perturbation. Most methods and tools created for preserving privacy fall into one or more of these categories and have some drawbacks.

2.3.1 Generalization and Suppression

Many researchers comment on how privatization algorithms can distort data. For example, consider privatization via generalization and suppression. Generalization can be done by replacing exact numeric values with intervals that cover a sub-range of values; e.g., 17 might become 15..20 or by replacing symbols with more general terms; e.g., "date of birth" becomes "month of birth". Suppression can be done by replacing exact values with symbols such as a *star* or a phrase like "don't know" [47].

According to Fung et al. [42], generalization and suppression, hide potentially important details in the quasi-identifiers that can confuse classification. Worse, these transforms may not guarantee

privacy.

Widely-used generalization and suppression approaches are used of privacy models such as *k*-anonymity, *l*-diversity, and, *t*-closeness. *K*-anonymity [34] makes each record in the table indistinguishable with *k-1* other records by suppression or generalization [34,48,49]. The limitations of *k*-anonymity, as listed by Brickell et al. [35] are many fold. They state that *k*-anonymity does not hide whether a given individual is in the database. Also, in theory, *k*-anonymity hides uniqueness (and hence identity) in a data set, thus reducing the certainty that an attacker has uncovered sensitive information. However, in practice, *k*-anonymity does not ensure privacy if the attacker has background knowledge of the domain. An example of *k*-anonymity in action and how background knowledge of an attacker can affect privacy is shown in Figure 2.1.

Machanavajjhala et al. [36] proposed *l*-diversity. The aim of *l*-diversity is to address the limitations of *k*-anonymity by requiring that for each QID group,[1] there are at least *l* distinct values for each sensitive attribute value. In this way an attacker is less likely to "guess" the sensitive attribute value of any member of a QID group.

Work by Li et al. [37], later showed that *l*-diversity was vulnerable to *skewness* and *similarity* attacks, making it insufficient to prevent attribute disclosure. Hence, Li et al. proposed *t*-closeness to address this problem. *t*-closeness focuses on keeping the distance between the distributions of a sensitive attribute in a QID group and that of the whole table no more than a threshold *t* apart. The intuition is that even if an attacker can locate the QID group of the target record, as long as the distribution of the sensitive attribute are similar to the distribution in the whole table, any knowledge gained by the attacker cannot be considered as a privacy breach because the information is already public. However, with *t*-closeness, information about the correlation between QIDs and sensitive attributes is limited [37] and so causes degradation of data utility.

[1] A QID group is a set of instances whose quasi-identifier values are the same because of generalization or suppression.

Consider the abbreviated *ant-1.3* data shown in the following table from the PROMISE data repository [50]. We assume that we want to share this data.

ID	QIDs								
name	wmc	dit	noc	cbo	rfc	lcom	ca	ce	loc^*
taskdefs.ExecuteOn	11	4	2	14	42	29	2	12	395
DefaultLogger	14	1	1	8	32	49	4	4	257
taskdefs.TaskOut- putStream	3	2	0	1	9	0	0	1	58
taskdefs.Cvs	12	3	0	12	37	32	0	12	310
taskdefs.Copyfile	6	3	0	4	21	1	0	4	136
types.Enumerated- Attribute	5	1	5	12	11	8	11	1	59
NoBannerLogger	4	2	0	3	16	0	0	3	59

The ten attributes of this table divide into two categories: *Identifier* ID and *Quasi-Identifiers* QIDs. Note that the sensitive QID is indicated by a superscript "*", in this case it's lines of code (loc). An ID could be anything that specifically identifies an individual or thing such as a social security number, first and last names or a filename. Unlike an ID, QIDs are not specific to a particular individual or thing. However, they can be used to re-identify an individual record in a database.

The first step in privatizing this data-set is to de-identify the table i.e., remove the identity attribute "name" (but note that for the ease of explanation, we leave the ID for the example). One might think that removing the ID column should be enough to protect individual privacy, however, research has shown that this is not the case [33,35,42,51]. In fact, using external public databases and/or personal background knowledge, an attacker can re-identify an individual record and associate that record with a sensitive attribute. For example, suppose the attacker has the following background knowledge.

$$rfc = 11 or lcom = 0$$

On the data with deleted IDs, this attacker might use this knowledge to select the rows containing *type.EnumeratedAttribute*, *taskdefs.TaskOutputStream*, and *NoBannerLogger*, thereby learning with 100% certainty that there are 58 or 59 lines of code in the target file.

Even after applying a privacy algorithm, that background knowledge can still be used to violate privacy. Suppose this attacker studies the following *k=2*-anonymous version of the above data:

ID	*QIDs*								
name	wmc	dit	noc	cbo	rfc	lcom	ca	ce	loc^*
taskdefs.ExecuteOn	11-14	<5	≤ 5	8-14	32-42	29-49	*	*	395
taskdefs.Cvs	11-14	<5	≤ 5	8-14	32-42	29-49	*	*	310
Default Logger	11-14	<5	≤ 5	8-14	32-42	29-49	*	*	257
taskdefs. TaskOut- putStream	<7	<5	≤ 5	1-4	*	≤ 8	0	≤ 4	58
taskdefs. Copyfile	<7	<5	≤ 5	1-4	*	≤ 8	0	≤ 4	136
types. Enumerated Attribute	<7	<5	≤ 5	*	11-16	≤ 8	*	≤ 4	59
NoBanner Logger	<7	<5	≤ 5	*	11-16	≤ 8	*	≤ 4	59
"*" denotes that any value is possible									

The background knowledge that $rfc = 11$ and $lcom = 0$ will result in 4 records being returned (the last four rows). With this result, because of the lack of diversity in the sensitive attribute of the result, an attacker will know with 75% certainty that the target has 58 or 59 lines of code.

Figure 2.1: Effects of background knowledge on privacy.

2.3.2 Bucketization

Bucketization arose as a way to handle high dimensional data, which cannot be handled by generalization. This is due to the "curse of dimensionality" [52]. Martin et al. [53] describes bucketization as follows:

1. Partition tuples or instances in a data set into "buckets";
2. Separate sensitive attributes from non-sensitive ones by randomly permuting the sensitive attribute values within each bucket.

The result is a sanitized data set that consists of the buckets with permuted sensitive values.

2.3.3 Anatomization and permutation

Anatomization and permutation both accomplish a similar task, that is, the de-association of the relationship between quasi-identifiers and sensitive attributes. However, anatomization does it by releasing "the data on QID and the data on the sensitive attribute in two separate tables..." with "one common attribute, *GroupID*" [42]. On the other hand, permutation de-associates the relationship between a QID and a numerical sensitive attribute. This is done by partitioning a set of data records into groups and shuffling the sensitive values within each group [54].

2.3.4 Perturbation

A precise definition of perturbation is put forward by Fung et al. [42]:

> *The general idea is to replace the original data values with some synthetic data values, so that the statistical information computed from the perturbed data does not differ significantly from the statistical information computed from the original data.*

It is important to note that the perturbed data records do not correspond to real world record owners. Also, methods used for perturbation include, additive noise, data swapping and synthetic

data generation. The drawback of these transforms is that they may not guarantee privacy. For example, suppose an attacker has access to multiple independent samples from the same distribution from which the original data was drawn. In that case, a principal component analysis could reconstruct the transform from the original to privatized data [55]. Here the attacker's goal is to estimate the matrix (M_T) used to transform the original data to it's privatized version. M_T is then used to undo the data perturbation applied to the original data. According to Giannella et al. [55], $M_T = WD_0Z'$, where W is the eigenvector matrix of the covariance matrix of the privatized data. Z' is the transform of the eigenvector matrix of the covariance matrix of the independent samples. Finally, D_o is an identity matrix.

2.3.5 Output Perturbation

Different to perturbation where the data is transformed to maintain its privacy, output perturbation maintains the original data in a database and instead adds noise to the output of a query [56]. Differential privacy falls into the category of *output perturbation*. Work by Dinur and Nissim [56] also fall into this category and forms the basis of Dwork's work on differential privacy [57, 58] (ε-differential).

According to Fung et al. [42], ε-differential privacy is based on the idea that the risk to the record owner's privacy should not substantially increase as a result of participating in a statistical database. So, instead of comparing the prior probability and the posterior probability before and after accessing the published data, Dwork et al. [57, 58] proposed to compare the risk with and without the record owner's data in the published data.

Dwork [57, 58] defines ε-differential privacy as follows:

> We say databases D1 and D2 differ in at most one element if one is a proper subset of the other and the larger database contains just one additional row.

Although ε-differential privacy assures record owners that they may submit their personal infor-

mation to the database securely, it does not prevent membership disclosure and sensitive attribute disclosure studied in this work. This is shown in an example from Chin and Klinefelter [59] in a Facebook advertiser case study. Through reverse-engineering, Chin and Klinefelter [59] inferred that Facebook uses differential privacy for its targeted advertising system. To illustrate the problem of membership and sensitive attribute disclosure, the authors described Jane's curiosity about her neighbor John's HIV status when she learned that he was on the finisher's list for the 2011 Asheville AIDS Walk and 5K Run. So armed with John's age and zip code, she went to Facebook's targeted advertising area and found that there was exactly one male Facebook user age 36 from zip code 27514 who listed the "2011 Asheville AIDS Walk and 5K Run" as an interest. At this point, Jane placed a targeted advertisement offering free information to HIV-positive patients about a new antiretroviral treatment. If charged by Facebook for having her ad clicked, Jane can assume with some level of certainty that John is HIV positive.

In practice, the above issues with privacy models and techniques are very real problems. Grechanik et al. [3] found that *k*-anonymity greatly degraded the test-coverage of data-centric applications. Furthermore, Brickell and Shmatikov [35] reported experiments where achieving privacy using the above methods "requires almost complete destruction of the data mining capability". They concluded that *depending on the privatization parameter*, the privatized data provided no additional utility vs. trivial privatization which privatizes data by simply removing all the sensitive attributes or all the other quasi-identifiers. Worse, they also reported that simplistic trivial privatization provides better privacy results than supposedly better methods like *l*-diversity, *t*-closeness and *k*-anonymity.

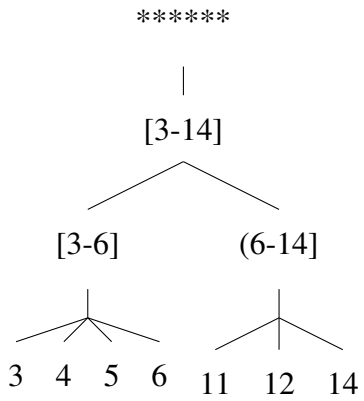
2.4 Privacy Algorithms

Fung et al. provided an excellent survey for privacy preserving data publishing [42]. In it they characterize 28 privacy algorithms. In this dissertation we focus on algorithms that change quasi-

identifiers of data in order to prevent sensitive attribute disclosure attacks. In the previous section we discussed some techniques that accomplish this, namely generalization and suppression, and perturbation. In this section, we will therefore focus on the algorithms that are based on these techniques.

2.4.1 Datafly for *k*-anonymity

Datafly [34, 60] uses generalization and suppression techniques to achieve the *k*-anonymity model of the quasi-identifiers of each target in a data set being indistinguishable from *k - 1* others. The core Datafly algorithm starts with the input of a set of quasi-identifiers, *k*, and a generalization hierarchy. An example of a hierarchy is shown in the tree below. Values at the leaves are generalized by replacing them with the sub-ranges [3-6] or (6-14]. These in turn can be replaced by [3-14]. Or the leaf values can be suppressed by replacing them with a symbol such as the *stars* at the top of the tree.



Datafly then replaces values in the quasi-identifiers according to the hierarchy. This generalization continues until there are *k* or fewer distinct instances. These instances are suppressed.

In software engineering, the Datafly algorithm for *k*-anonymity has been explored by Grechanik et al. [3]. Therefore we use it in this dissertation as a benchmark for CLIFF&MORPH, the privacy algorithm included in LACE.

2.4.2 Incognito for *k*-anonymity

Incognito is a suite of optimal bottom-up generalization algorithms presented by LeFevre et al. [42, 61, 62], to generate all possible k-anonymous full-domain generalizations. According to Ciriani et al. [62], Incognito follows a bottom-up breadth-first search strategy on the domain generalization hierarchy. They state that first (iteration 1), Incognito checks *k*-anonymity for each single quasi-identifier, discarding those generalizations that do not satisfy *k*-anonymity for the single attribute. Then, it combines the remaining generalizations in pairs performing the same control on pairs of attributes (iteration 2); then in triples (iteration 3), and so on, until all the quasi-identifiers are considered. In other words, for each combination, Incognito checks the satisfaction of the *k*-anonymity constraint with a bottom-up approach; when a generalization satisfies *k*-anonymity, all its direct generalizations also certainly satisfy *k*-anonymity and therefore the algorithm terminates.

2.4.3 *PriestPrivacy* for Data Swapping

In software engineering, Taneja et al. [5] use *data swapping* to privatize databases for database-centric applications. Their algorithm is called *PriestPrivacy*. Data swapping is a standard perturbation technique used for privacy [5, 42, 63]. This is a permutation approach which preserves the original values of data as it de-associates the relationship between a non-sensitive quasi-identifier and a numerical sensitive attribute.

As described by Taneja et al. [5], the *PriestPrivacy* algorithm takes as its inputs the matrix of quasi-identifiers and their values, and the value of the probability that the original data will remain unchanged in the privatized data set. The output is a privatized matrix that has the same dimensions and semantics as the quasi-identifier matrix.

To get this privatized matrix, *PriestPrivacy* iterates through the attributes in the quasi-identifier matrix computing the distinct values for each quasi-identifier. Next, the algorithm iterates through all the rows for the given quasi-identifier and randomly replaces the original value in a cell with

one of the original distinct values for the given quasi-identifier. We also use *PriestPrivacy* in this dissertation as a benchmark for CLIFF&MORPH.

2.5 Evaluating Privacy

Three privacy evaluation methods are outlined in work by Torra et al. [64]. These measures are 1) Information loss measures, 2) Disclosure risk measures and 3) Scores. Information loss measures are designed to establish in which extent published data is still valid for carrying out the experiments planned on the original data. They take into account the similarity between the original data set and the protected one, as well as the differences between the results that would be obtained with the original data set and the results that would be obtained from the disguised data set. Torra et al. [64] further explains that disclosure risk measures are used to evaluate the extent in which the protected data ensures privacy and that *scores*, is a summary both information loss and disclosure risk, that is, when these two measures are commensurate, it is possible just to combine them using the average.

2.5.1 Privacy Metrics

Privacy is not a binary step function where something is either 100% private or 100% disclosed. Rather it is a probabilistic process where we strive to decrease the likelihood that an attacker can uncover something that they should not know. The rest of this section reviews some privacy metrics used in testing and debugging research and finally defines privacy using a probabilistic *increased privacy ratio* (IPR), of privatized data sets.

Privacy metrics allows you to know how private the disguised version of your data is. There are two categories of privacy metrics in the literature: syntactic and semantic [35]. The syntactic measures considers the distribution of attribute values in the privatized data set and are used by algorithms such as *k*-anonymity and *l*-diversity. In comparison, the semantic metrics measures

what an attacker may learn or the incremental gain in knowledge caused by the privatized data set [35] and use distance measures such as the Earth Movers Distance, KL-divergence and JS-divergence to quantify this difference in the attackers knowledge. Other methods in the software engineering literature include *Increased Privacy Ratio (IPR)* [11, 12], *entropy* [7, 8] and *guessing anonymity* [5, 6, 65].

Previous privacy studies in software engineering [7, 8] have used entropy to measure privacy. Entropy (H) measures the level of uncertainty an attacker will have in trying to associate a target to a sensitive attribute. For example, if querying a data set produces two instances with the same sensitive attribute value then the attacker's uncertainty level or entropy is zero. However, if the sensitive attribute values are different, the attacker has a $\frac{1}{2}$ chance of associating the target with the correct sensitive attribute value. The entropy here is 1 bit, assuming that there are only two possible sensitive values. In general, the entropy of a data set is, $H = \sum_{i=1}^{|S|} p(s_i)|log_2 p(s_i)|$ bits which corresponds to the number of bits needed to describe the outcome. Here S is the set of sensitive attribute values and s_i is the probability that $S = s_i$.

Guessing anonymity was introduced by Rachlin et al. [65] and it is describe as a privacy definition for noise perturbation methods. Guessing anonymity of a privatized record in a data set is the number of guesses that the optimal guessing strategy of the attacker requires to correctly guess the record used to generate the privatized record [5, 6, 65].

We introduced Increased Privacy Ratio (IPR) in previous work [11]. It is based on the *adversarial accuracy gain,* A_{acc} from the work of Brickell and Shamtikov [35]. According to the authors definition of A_{acc}, it quantifies an attacker's ability to predict the sensitive attribute value of a target t. A_{acc} measures the increase in the attacker's accuracy after he observes a privatized data set and compares it to the baseline from a trivially privatized data set which offers perfect privacy by removing either all sensitive attribute values or all the other QIDs.

2.6 Privacy for Testing and Debugging

LACE is designed based on the privacy needs of cross project defect prediction. Other researchers in SE focus on privacy in software testing and debugging [4–8], This becomes an issue when it involves:

- Collecting user information after a software system has been deployed [7, 8];
- Or outsourcing the software testing to third parties (e.g. see Budi et al. [4], Taneja et al. [5] and Li et al [6]). In this case, companies do not wish to release actual cases for testing. Hence, they anonymize the test cases before releasing them to testers.

Work published by Castro et al. in 2008 [7], sought to provide a solution to the problem of software vendors who need to include sensitive user information in error reports in order to reproduce a bug. To protect sensitive user information, Castro et al. [7] used symbolic execution along the path followed by a failed execution to compute path conditions. Their goal was to compute new input values unrelated to the original input. These new input values satisfied the path conditions required to make the software follow the same execution path until it failed.

As a follow-up to the Castro et al. [7] paper, Clause et al. [8] presented an algorithm which anonymized input sent from users to developers for debugging. Like Castro et al. [7], the aim of Clause et al. was to supply the developer with anonymized input which causes the same failure as the original input. To accomplish this, they first used a novel "path condition relaxation" technique to relax the constraints in path conditions thereby increasing the number of solutions for computed conditions.

In contrast to the work done Castro [7] and Clause [8], Taneja et al. [5] proposed PRIEST, a privacy framework. Unlike our work, which privatizes data randomly within "nearest unlike neighbor" border constraints, the privacy algorithm in PRIEST is based on *data-swapping* where each value in a data set is replaced by another distinct value of the same attribute. This is done according to some probability that the original value will remain unchanged.

Work by Taneja et al. [5], followed work done by Budi et al. [4]. Similarly, their work focused on providing privatized data for testing and debugging. They were able to accomplish this with a novel privacy algorithm called *kb*-anonymity. This algorithm combined *k*-anonymity with the concept of program behavior preservation which guide the generation of new test cases based on known ones and make sure the new test cases satisfy certain properties [4]. The difference with the follow-up work by Taneja et al [5], is that while Budi et al. [4] replaces the original data with new data, in Taneja's work [5], the data-swapping algorithm maintains the original data and offers individual privacy by swapping values.

Software test outsourcing work by Li et at. [6], follows a similar approach to our work in privacy for cross project defect prediction (CLIFF&MORPH [12] and now LACE): **1)** Don't use all the data (minimize), and **2)** obfuscate data that are used. Li et al. accomplish this through the process of securing centroids using a novel combination of data mining approaches, program analysis, and privacy constraints.

2.7 Summary

Research in privacy preserving data publishing focuses on accomplishing two goals, i.e., the privacy and utility of the data to be shared (published). The privacy algorithms that make up LACE (CLIFF&MORPH) are designed to balance these goals via data minimization with CLIFF and constrained obfuscation with MORPH. This is different from the earlier methods like *k*-anonymity, *l*-diversity and *t*-closeness which only focus on offering privacy via obfuscation of data to decrease the likelihood of re-identification disclosure and sensitive attribute disclosure discussed in Section 2.1. In terms of utility, research concerning the application of *k*-anonymity to software testing was shown to be unsuccessful as privacy was increased (by increasing *k*) [3]. Since neither *l*-diversity nor *t*-closeness take utility into consideration, we conjecture that they would also produce weak utility results. More recent work in software testing focuses on finding that balance between

privacy and utility by using data swapping and data minimization [5, 6].

There are many aspects of privacy research to take into account when seeking this balance between privacy and utility, including privacy threats, models, algorithms and metrics used to evaluate these algorithms. As privacy research continues to grow in software engineering, we apply these aspects of privacy research to cross project defect prediction.

Chapter 3

Software Defect Prediction

3.1 Introduction

All the experiments in this dissertation focuses on defect prediction. This chapter describes that kind of prediction, why we explore it and what is the missing in the current literature on defect prediction (the need to privatize data before it is shared).

According to Lessmann et al. [66], the goal of software defect prediction is to improve software quality and testing efficiency via predictive classification models to allow for the timely identification of defective modules. Cross project defect prediction is recognized as a type of software defect prediction that deals with the challenge of those organizations with insufficient data to build their own quality defect predictors. It is an active field of study whose main component involves data sharing [18, 19, 24, 27, 28]. Many studies exist which focus on improving the quality of cross project defect predictors, however very few consider the privacy concerns that arise from needing access to *other* data [11, 12]. Failure to address the privacy concerns of data owners can stall this field of research.

In this chapter we present a brief review of the field of the economics of software defect prediction, static code defect prediction since all the data used in our experiments are static code metrics.

We then elaborate on cross project defect prediction and transfer learning techniques used to improve on the defect predictors built with *other* data. Finally, we conclude this chapter with the empirical study of He&Peters et al. [28], showing that with transfer learning open-source data can predict for proprietary data.

3.2 Software Defect Prediction Economics

Many researchers have documented the economical value of early defect detection [1, 67, 68]. Boehm & Papaccio advised that reworking software (e.g. to fix bugs) is cheaper earlier in the life cycle than later "*by factors of 50 to 200*" [67]. Other research make the same conclusion. A panel at IEEE Metrics 2002 concluded that finding and fixing severe software problems after delivery is often 100 times more expensive than finding and fixing them during the requirements and design phase [68].

Finally, Dabney et al. [1] studied the cost of quickly fixing an issue relative to leaving it for a later phase (data from four NASA projects - see Figure 3.1). He found that delaying issue resolution even by one phase increases the cost-to-fix to $\Delta = 2 \ldots 5$. Using this data, Dabney et al. [1] calculated that a dollar spent on verification returns to NASA, on those four projects, $1.21, $1.59, $5.53, and $10.10, respectively.

Once we accept that it is economically effective to find bugs earlier, then the next question becomes "how do we find the bugs earlier?" Defect prediction learned from static code measures allows software companies to take advantage of early defect detection [69, 70]. Models made for defect prediction are usually built with local or within-company data sets using common machine learners. The data sets are comprised of independent variables such as the *code metrics* used in this work and one dependent variable or prediction target with values (labels) to indicate if defects are present.

		Phase issue found					
	Phase issue	f=1	f=2	f=3	f=4	f=5	f=6
i	introduced	Requirements	Design	Code	Test	Int	Ops
1	Requirements	1	5	10	50	130	368
2	Design		1	2	10	26	74
3	Code			1	5	13	37
4	Test				1	3	7
5	Integration					1	3
Δ = mean $\left(\frac{C[f,i]}{C[f,i-1]}\right)$			5	2	5	2.7	2.8

Figure 3.1: Cost-to-fix escalation factors. From [1]. Here, $C[f,i]$ denotes the cost-to-fix escalation factor relative to fixing an issue in the phase where it was found (f) versus the phase where it was introduced (i). The last row shows the cost-to-fix delta if the issue introduced in phase i is fixed immediately afterward in phase $f = i+1$.

3.3 Static Code Defect Prediction

A typical, object-oriented, software project can contain hundreds to thousands of classes. In order to guarantee general and project-related fitness attributes for those classes, it is commonplace to apply some quality assurance (QA) techniques to assess the classes's inherent quality. These techniques include inspections, unit tests, static source code analyzers, etc. A record of the results of this QA is a defect log. We can use these logs to learn defect predictors, if the information contained in the data provides not only a precise account of the encountered faults (i.e., the "bugs"), but also a thorough description of static code features such as Lines of Code (LOC), complexity measures (e.g., McCabe's cyclomatic complexity [71]), and other suitable object-oriented design metrics [72–74].

For this, data miners can learn a predictor for the number of defective classes from past projects so that it can be applied for QA assessment in future projects. Such a predictor allows focusing the QA budgets on where it might be most cost effective. This is an important task as, during development, developers have to skew their quality assurance activities towards artifacts they believe require most effort due to limited project resources.

Now, static code defect predictors yield a lightweight sampling policy that, based on suit-

able static code measures, can effectively guide the exploration of a system and raises an alert on sections that appear problematic. One reason to favor static code measures is that they can be automatically extracted from the code base, with very little effort even for very large software systems [75]. The industrial experience is that defect prediction scales well to a commercial context. Defect predicting technology has been commercialized in Predictive [76], a product suite to analyze and predict defects in software projects. One company used it to manage the safety critical software for a fighter aircraft (the software controlled a lithium ion battery, which can over-charge and possibly explode). After applying a more expensive tool for structural code coverage, the company ran Predictive on the same code base. Predictive produced results consistent with the more expensive tool. But, Predictive was able to faster process a larger code base than the more expensive tool [76].

In addition, defect predictors developed at NASA [69, 70] have also been used in software development companies outside the US (in Turkey). When the inspection teams focused on the modules that trigger the defect predictors, they found up to 70% of the defects using just 40% of their QA effort (measured in staff hours) [77].

Finally, a subsequent study on the Turkish software compared how much code needs to be inspected using random selection versus selection via defect predictors. Using random testing, 87% of the files would have to be inspected in order to detect 87% of the defects. However, if the inspection process was restricted to the 25% of the files that trigger the defect predictors, then 88% of the defects could be found. That is, the same level of defect detection (after inspection) can be achieved using $\frac{87-25}{87} = 71\%$ less effort [78].

3.4 CPDP = Cross Project Defect Prediction

When data can be shared between organizations, such as the NASA and Turkey study mentioned in the previous section, defect predictors from one organization can generalize to another. However,

initial experiments with cross-project learning were either very negative [19] or inconclusive [17]. Zimmermann et. al. [19] observed, defect prediction via local data is not always available to many software companies as

- The companies may be too small.
- The product might be in its first release and so there is no past data.

Kitchenham et al. [17], who studied cross versus within-company cost estimation, saw problems with relying on *within-company* data sets. They noted that the time required to collect enough data on past projects from a single company may be prohibitive. Additionally, collecting *within-company* data may take so long that technologies used by the company would have changed and therefore older projects may no longer represent current practices.

Recently, we have had more success using better selection tools for training data [18, 25] but this success was only possible if the learner had unrestricted access to all the data. For example, let us return to the results of the NASA and Turkey study. The defect predictors developed at NASA [69, 70] were used in software development companies in Turkey. Inspection teams found up to 70% of the defects using just 40% of their QA effort (measured in staff hours) [77]. Work by Rahman et al. [20], also show the success of CPDP. Their focus was on cost sensitive prediction where only the top *n*% of reported defect prone lines or files were used in CPDP and within (local) experiments.

3.5 Measuring the Feasibility of CPDP

Studies in CPDP benchmark their work against within (local) defect prediction [18]. However, recent studies have shown that the success of CPDP depends on selecting or creating the right training instances and so the feasibility of CPDP should depend on the number of test sets able to access a *training instances+predictor* combination that will build a defect model whose performance meet

user criteria [19, 24].

More recent work in CPDP, have avoided the within (local) comparison for judging the success of CPDP. Instead CPDP success is based on perceived user criteria and conclusions are made based on these results. For instance, Zimmermann et al. [19] built 622 cross-company predictions and found that only 3.4% met the criteria they used to determine if a project was a strong predictor for another project (accuracy, precision and recall were above 75%). From these results, they concluded that CPDP remained a challenge. However, they also studied the factors which influence the the success of CPDP and used these factors to derive decision trees that provided estimates for precision, recall, and accuracy before a prediction was attempted. In others words careful selection of training data can determine the success of CPDP.

Similar to the work done by Zimmermann et al. [19], He et al. [24] checked when cross project defect predictions that were successful. In their work, defect predictors were considered strong if recall was above 70% and precision above 50%. Although their criteria was different and less stringent than the Zimmermann et al. study [19], their results were similar, ranging from 0.32% to 4.67% of cross-company predictions that met their criteria over five different learners. However, the authors did not use this result as a measure of the success of CPDP, instead they based their measure on the results of selecting the best training instances for a test set. On average their results met their criteria. Out of 34 test sets 18 were considered as strong defect predictors. From those results, they concluded that CPDP was feasible as long as it involved the careful selection of training data. To help with the selection of training data, we look to the field of transfer learning.

3.6 Transfer Learning

Researchers of cross project defect prediction have turned to transfer learning to improve the quality of defect predictors built from *other* data. Transfer learning techniques therefore assume easy access to data which may not always be the case. As a result, it is increasingly necessary that

research also focus on privacy concerns of data owners. In this way, with access to privatized data, researchers can continue to add to this field of research.

In this section we report on the literature for transfer learning starting with machine learning and then exploring techniques used in cross project defect prediction.

In the machine learning literature, the 2010 article by Pan&Yang [79] is the definitive definition of transfer learning. Pan&Yang state that transfer learning is defined over a *domain D*, which is composed of pairs of examples X and a probability distribution about those examples $P(X)$; i.e., $D = \{X, P(X)\}$. This P distribution represents what class values to expect, given the X values.

The transfer learning *task* T is to learn a function f that predicts labels Y; i.e., $T = \{Y, f\}$. Given a new example x, the intent is that the function can produce a correct label $y \in Y$; i.e., $y = f(x)$ and $x \in X$. According to Pan&Yang, synonyms for transfer learning include, learning to learn, life-long learning, knowledge transfer, inductive transfer, multitask learning, knowledge consolidation, context-sensitive learning, knowledge-based inductive bias, metalearning, and incremental/cumulative learning.

Transfer occurs between a source domain D_s and a target domain D_t. Transfer might be required since there is not enough data in the target to learn a predictive function (in which case $|D_t| \ll |D_s|$). If transfer learning is successful, then we can find a function in the target domain f_t that generates correct labels in the target domain. This function f_t can be used by management to create and justify their decisions. That is, the above is a formal definition of the process of finding and reusing best practices. For example, Company X's source data indicates that large C/C++ files have more defects than small or non C/C++ files. Thus a function or *rule* is learned that associates "large" and "C/C++" files with the label "more defect-prone". Best practice would then be to re-factor large "C/C++" files to reduce defects.

If transfer learning fails, this may be due to *negative transfer* where the source domain data contributes to reduced performance in the target domain [79]. This can be caused by:

- The source and target are too dissimilar; or

- The wrong data are moved from source to target.

Note that there may be no solution to dissimilarity. However, in the latter case, negative transfer can be avoided using a *relevancy filter* that carefully selects what is transferred [18]. One such filter (discussed in Section 3.6.1) is the Turhan filter that generates training sets by rejecting data that is far away from the test sets.

3.6.1 Instance-Transfer

Inspired by Kitchenham et al.'s 2007 TSE article "Cross- vs. Within-Company Cost Estimation Studies" [17], Menzies et al. tried transferring defect models between American NASA software (for ground systems and flight missions) and Turkish software controllers (for whitegoods such as ovens and refrigerators). The initial results were not promising:

- Round-robin experiments were conducted with 10 data sets: models were learned on nine and tested on the tenth.
- The resulting defect prediction models exhibited high recalls (on average, 94%).
- But the false alarm rates were unsatisfactory (over 67%).

A moment's reflection explains the above results. If we walk into a very large crowd asking "am I defective?" sooner or later some oracle will say "yes". However, in a large crowd, many oracles can be wrong. Hence, we should expect that large crowds generate high recalls with high false alarms.

While unaware of the Pan&Yang's paper, Turhan's fix [18] to the above problem can be expressed as "avoid negative transfer". Turhan reasoned that the high false alarms were a result of using too many unrelated projects. Hence, he proposed the following *relevancy filter*:

- For each test item in the target, find the $k = 10$ nearest neighbors in the source domain (in

the case of the round-robin experiments, the "source" was the union of nine data sets and the "target" was the tenth data set).

- Using an off-the-shelf Naive Bayes classifier, learn a defect predictor from a training set formed from the union of all those $k = 10$ neighbors.

The models formed in this way had recall and false alarm rates of (on average) $(69, 27)\%$ respectively- a performance level that is (a) competitive with defect predictors learned from just the target domain; (b) much larger than known performance levels for humans using manual methods[1]; and (c) sufficient to guide inspection teams to code sections with the most errors[2].

In terms of the Pan&Yang taxonomy, the work of Turhan et al. [18] is an *instance-transfer* method since it samples training data according to a weight function that returns either one (if that instance is within the k=10 nearest neighbors of a test instance) or zero (otherwise).

Turhan et al. [18] used a simple Euclidean measure over i static code measures seen in source and target instances x_s, x_t: $d = \sqrt{\sum_i (x_s^i - x_t^i)^2}$. Ma et al. [21] repeated parts of the Turhan et al. study using the same data sets [18][3], but with a different distance measure. For each feature, they assigned the values $\{0,1\}$ to source domain ranges if they overlapped with the min,max range seen in the target ranges. Then, the distance s_i between a source and target instance was the sum of the ranges that overlapped. Finally, appealing to gravitational physics, they assigned weights proportional to $\frac{1}{s_i^2}$. These weights were then used to adjust the weights on frequency counts within a Naive Bayes classifier (their justification for this approach was to ignore examples that are nearest, but remote).

The Turhan & Ma et al. studies use *fine-grain* relevancy filters. In the fine-grain approach, given a table of X instances, each instance might be used with a different weight. An alternate

[1] Humans can find up to $\approx$ 20%..60% of the bugs [69].

[2] Weyuker et al. report that such predictors can isolate 20% of the code with 80% of the errors [80]. Tosun et al. report that software inspection that focus on code that triggers these predictors find up to 70% of the defects using just 40% of their QA original effort (measured in staff hours) [77].

[3] All this data is available in the PROMISE repository http://promisedata.googlecode.com.

approach, explored by He et al. [24, 28], is to perform *coarse-grain* relevancy at the table level. In this approach, all instances in one table get the same weight and, often, most tables get a zero weight for all instances.

He et al. [24, 28] have defined two coarse-grained filters (the second is less computationally expensive than the first, and has elements of feature representation transfer and parameter transfer). *CoarseFilter1* [24] was applied to a *corpus* of 34 defect data sets from the PROMISE repository. For every $one \in corpus$ data sets, explore triplets of size three $(x,y,z) \in \{corpus - one\}$. Each combination of (x,y,z) data sets was tested on the other 30 data sets ($\{corpus - one - x - y - z\}$). The highest scoring triplet was then selected as the training set.

CoarseFilter2 [28] was used on a different *corpus* of 34 releases of 10 open source data sets and 34 releases of 7 proprietary data sets. For each $test \in corpus$, the *train* set were the ten "closest" data sets, calculated as follows:

- Let $other = corpus - test$.
- For each $one \in other$, select K instances at random from *test* and *one* (so $2K$ instances in all).
- The distance between *one* and *test* is the error rate seen in predictions if a model is learned from the $2K$ instances and applied back to itself (zero error = zero distance).

He et al. found that with *CoarseFilter1* in half of their experiments, cross-company learning using coarse-grain relevancy filtering can generate defects models just by learning from the local within-data [24]. Also, using *CoarseFilter2*, He&Peters et al. [28] found that it was possible to successfully learn a defect predictor for open source projects, then apply it to proprietary projects [28].

Like Turhan et al. [18], we characterize the work of Ma&He et al as *instance-transfer* methods that propose a range of weighting schemes for instances.

As research continues in transfer learning techniques for cross project defect prediction, as seen in the example of the next section, any future research in this field can be severely inhibited

by privacy. The work in this dissertation seeks to avoid that. We expand on the He&Peters et al. study in the following section [28].

3.7 Open-Source Predicts for Projects

With the found success of applying transfer learning techniques to CPDP, research has now expanded to building defect predictors from open-source data to predict for proprietary data [28]. In the following section we share the techniques, and experimental results from the He&Peters et al. [28] empirical study.

Most of the research done in cross project defect prediction only evaluate open source data [11, 12,21,24]. Those that included proprietary data in their experiments maintained that not only could these *data not be shared*, but that cross defect prediction was a challenge between open source and proprietary data [19]. Specifically, in an example of cross defect prediction between Firefox and Internet Explorer, Zimmermann et al. [19] found that while Firefox defect data could build a strong defect predictor for Internet Explorer, the reverse was not true. In other words, Firefox (the open source project) predicts for Internet Explorer, but Internet Explorer (the proprietary project) could not predict Firefox. This empirical study further explores the notion of open source projects predicting for proprietary projects.

Considering the differences between development environment of open-source projects and proprietary projects, it would be risky to directly apply knowledge of open-source projects to proprietary projects. For example, a widely cited comparison study conducted by Zimmermann et al. shows that cross project defect prediction rarely succeeded [19]. Hence the following question:

Can we transfer defect prediction knowledge from open-source projects to proprietary projects?

We argue that learning from open-source projects is a practical solution for proprietary projects lacking local historical data because:

- The measurement data of open-source projects is easy and cheap to obtain;

- Companies are usually not willing to share their proprietary data because of the risk of privacy and safety [12].

3.7.1 Methodology

Data

In this study we investigate a total number of 68 datasets (34 releases of 10 open-source projects and 34 releases of seven proprietary projects) shown in Table 3.1 with attribute details in Table 2.1. These data sets were collected and shared by Jureczko et al. [15, 81]. Each instance in a data set represents a Java class of the release and consists of two parts: instance features including 20 static code metrics and a labeled feature "bug" indicating how many defects in that class. In this study, we consider a class as defect-free (DF) if the value of bug is equal to 0; otherwise, defect-prone (DP). The goal of defect prediction in this study is to identify defect-prone classes precisely. These data sets are publicly accessible and have been used in many previous studies, which make it easy to repeat and compare our experiments to other related studies.

Learning Algorithms

In this study we use three learners widely used in defect prediction research. First, Random Forest (RF) is selected for its good performance. A benchmarking study done by Lessmann et al. showed that RF performs significantly better than 21 other predictors [66]. Second, we select Naïve Bayes according to recommendation of Menzies et al. which reported that the simple NB learner performs as well as complex learners [69]. Third, Logistic Regression (LR) which is favored by Zimmermann et al. [19].

1) **Random Forest, RF** is a collection of decision trees where each tree is learned from a bootstrap sample (randomly sampling the data with replacement) [82]. The features used to find the best split at each node is a randomly chosen subset of the total number of features. Each tree in

Table 3.1: Objective Data Sets

Open-source project data (training sets)				**Proprietary project data (test sets)**			
Dataset	**Size**	**#DP**	**%DP**	**Dataset**	**Size**	**#DP**	**%DP**
ant-1.3	187	20	10.7	prop1-ver128	3619	220	6.1
ant-1.4	265	40	15.1	prop1-ver164	3541	319	9
ant-1.5	401	32	8	prop1-ver192	3692	85	2.3
ant-1.6	523	92	17.6	prop1-ver44	4081	376	9.2
ant-1.7	1066	166	15.6	prop1-ver9	4455	149	3.3
camel-1	436	13	3	prop1-ver92	3670	1287	35.1
camel-1.2	765	216	28.2	prop2-ver225	1864	147	7.9
camel-1.4	1122	145	12.9	prop2-ver236	2403	76	3.2
camel-1.6	1252	188	15	prop2-ver245	2023	103	5.1
ivy-1.1	135	63	46.7	prop2-ver256	2025	625	30.9
ivy-1.4	321	16	5	prop2-ver265	2372	229	9.7
ivy-2	477	40	8.4	prop2-ver276	2472	334	13.5
jEdit-3.2.1	508	90	17.7	prop3-ver285	1709	177	10.4
jEdit-4	606	75	12.4	prop3-ver292	2330	209	9
jEdit-4.1	644	79	12.3	prop3-ver305	2388	89	3.7
lucene-2	288	91	31.6	prop3-ver318	2440	365	15
lucene-2.2	381	144	37.8	prop4-ver347	2906	162	5.6
lucene-2.4	536	203	37.9	prop4-ver355	2802	924	33
poi-1.5	300	141	47	prop4-ver362	2865	213	7.4
poi-2.0RC1	386	37	9.6	prop42-ver452	317	33	10.4
poi-2.5.1	466	248	53.2	prop42-ver453	259	20	7.7
poi-3	531	281	52.9	prop42-ver454	295	13	4.4
synapse-1	162	16	9.9	prop43-ver461	1730	32	1.9
synapse-1.1	230	60	26.1	prop43-ver472	1740	32	1.8
synapse-1.2	269	86	32	prop43-ver481	1884	33	1.8
velocity-1.4	224	147	65.6	prop43-ver492	1888	27	1.4
velocity-1.5	246	142	57.7	prop43-ver501	2172	83	3.8
velocity-1.6.1	261	78	29.9	prop43-ver512	2265	134	5.9
xalan-2.4.0	862	110	12.8	prop5-ver4	3514	264	7.5
xalan-2.5.0	945	387	41	prop5-ver40	3815	466	12.2
xalan-2.6.0	1170	411	35.1	prop5-ver85	3509	930	26.5
xerces-init	206	77	37.4	prop5-ver121	3445	425	12.3
xerces-1.2.0	515	71	13.8	prop5-ver157	2863	367	12.8
xerces-1.3.0	545	69	12.7	prop5-ver185	3260	268	8.2
mean	507	120	25.7	mean	2547	271	9.9
median	451	88	17.7	median	2421	193	7.8

the collection is used to classify a new instance. The forest then selects a classification by choosing the majority result.

2) **Naïve Bayes, NB** is a statistical learning algorithm that assumes features are equally important and statistically independent. The NB learner builds its classification power based on the Baye's rule shown below [83]:

$$P(c_k|x) = P(c_k) \times \frac{P(x|c_k)}{P(x)}$$

where c_k is a member of the set of values for the dependent attribute. In our case c_k could be defect-prone or defect-free. Also, x represents a test instance or unknown instance. So, to classify a test instance, NB finds the conditional probability of that instance being labeled c_k. The c_k with the highest probability is chosen as the label for x.

3) **Logistic Regression, LR** Afzal recommended LR when the dependent variable is dichotomous [84]. LR avoids the Gaussian assumption used in standard Naïve Bayes. The form of the logistic regression model is:

$$log(\frac{p}{1-p}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_k X_k$$

where p is the probability that the fault was found in the module and $X_1, X_2, ..., X_k$ are the independent variables. $\beta_0, \beta_1, ..., \beta_k$ are the regression coefficients estimated using maximum likelihood.

Data Preprocessing

In our experiments we adopt some data preprocessing operations to address quality problems of the experimental data.

- Data sampling: in each prediction we applied under-sampling to training data to reduce the effects of imbalance distribution (i.e., the ratio of defect-free classes is usually much higher than that of defect-prone classes). During our experiments we found that under-

sampling helps to improve prediction performance, which is consistent with what Menzies et al. found [85].

- Normalization: we normalize each attribute to the 0-1 intervals to reduce the effect of different value scales.

- Discretization: for predictions using the Naïve Bayes learner, we convert continuous attributes into nominal values using 5 equal-frequency bins. The reason we adopted equal-frequency is that distributions of most attributes are seriously skewed in our experimental data sets. For learners that can handle numerical attributes (i.e., Random Forest, Logistic Regression), we input the original data without discretization operation.

Performance Assessment

In the context of defect prediction, defect-prone classes are usually considered as positive instances. Consequently the 4 categories of defect prediction results are defined as below:

- *TP (True Positive)*: defect-prone classes that are classified correctly;

- *FN (False Negative)*: defect-prone classes that are wrongly classified to be defect-free;

- *TN (True Negative)*: defect-free classes that are classified correctly;

- *FP (False Positive)*: defect-free classes that are wrongly classified to be defect- prone.

In this study we employ probability of defects (PD), probability of false alarm (PF), and the G-measure which is the harmonic mean of PD and 1-PF, to assess performance of defect prediction models. There is a debate on right metrics to use when evaluating software defect prediction results [86], we do not explore this as it is out of the scope of this chapter. Instead, we adopt these three indicators based on the widely cited studies done by Menzies et al. [69, 70]. According to

Table 3.2: Some popular measures used in software defect prediction work.

		Actual	
		DP	**DF**
Predicted	**DP**	TP	FP
	DF	FN	TN
PD	$\frac{TP}{TP+FN}$		
PF	$\frac{FP}{FP+TN}$		
G	$\frac{\textbf{2*pd*(1-pf)}}{\textbf{pd+(1-pf)}}$		

definitions of these three indicators, we favor prediction results with high PD, low PF, and high G-measure. Table 3.2 shows how we calculate the three measures used in this work.

To eliminate the potential bias caused by data sampling, we repeated each prediction 10 times and use the median performance as the overall prediction performance.

3.7.2 Research Method

We conduct empirical studies on aforementioned objective data sets in Table 3.1. Our empirical studies include:

- A simulation experiment to verify the feasibility of using open-source project data to build defect prediction models for proprietary projects.
- A empirical study to verify the effectiveness and computational cost of a new proposed training data selection method. The idea behind our proposed data selection method is to select potential training data based on data similarity. More specifically, we first find cross training sets that are close to the test set, then we find features that cause the differences between training set and test set (i.e., unstable features), finally we remove these unstable features and learn cross project defect prediction models from remaining data of close training sets

The following two sub-sections describe details of these two empirical studies.

Usability of Open-Source Project Data

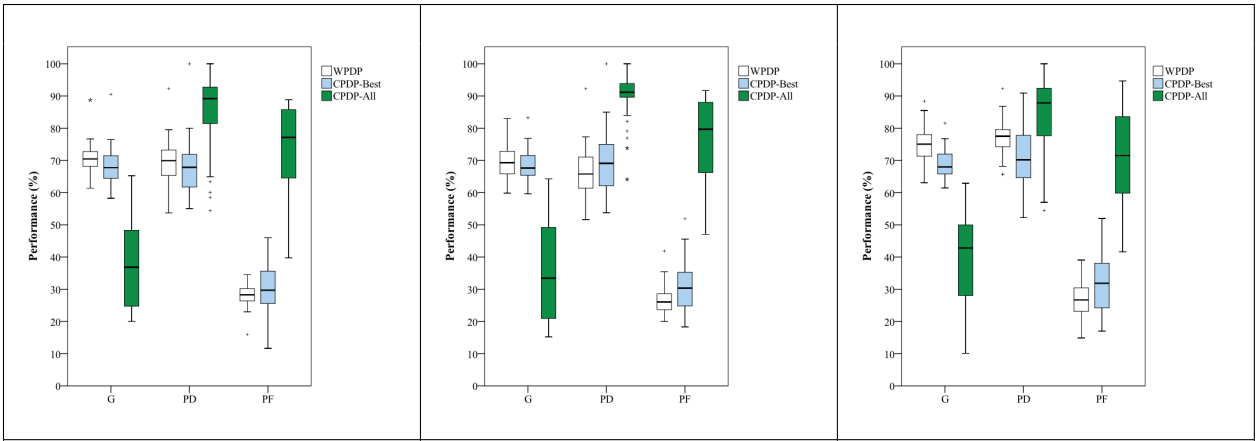
Before employing open-source project data to build defect prediction models for proprietary projects, we need to assess their usability for model training, i.e. we need to find evidence showing that learning from open-source project is feasible. To achieve this goal we conduct a simple simulation experiment as follows:

- *Cross project defect prediction using the most appropriate training set (CPDP-Best).* For each test set (i.e., a release of proprietary projects) we train defect prediction models on each individual cross project training set (i.e., a release of open-source projects) and observe its performance on the test set. We then select the best prediction performance for each test set. The results from this simple simulation provides empirical evidence indicating whether cross project defect prediction is feasible.

We have to point out that CPDP-Best is a post-facto approach and it does not work for practical prediction settings. We use CPDP-Best here as an baseline to explore the theoretical possibility of learning from open-source projects. We compared prediction performance of CPDP-Best with those of two other baselines described below:

- *Within project defect prediction (WPDP).* This baseline represents prediction performance of a within project learning scenario, i.e., 5-folds cross validation on the test set. Theoretically, performance of the within project defect prediction is the best performance we can achieve since in this scenario the training data and the test data are sampled from the same distribution.

- *Cross project defect prediction using all available training data (CPDP-All).* This baseline represents a very intuitive cross project defect prediction setting: combine all open-source



(a) Naïve Bayes (b) Logistic Regression (c) Random Forest

Figure 3.2: Comparisons of prediction performance among CPDP-Best, WPDP, and CPDP-All.

project data into a big training set without any data selection or filtering operation, and then learn prediction models from it. This cross project defect prediction would have success only if all open-source projects and proprietary projects hold the same global pattern for defect prediction. Turhan et al. found that CPDP-All would generate prediction results with very high PF, which was the main reason that motivated the data selection research for cross project defect prediction [18].

Figure 3.2 illustrates the comparisons of prediction performance. Note that in this simulation experiment, prediction results of CPDP-Best refer to results that have the greatest G-measure values. Intuitively we can see that the performance of CPDP-Best is close to that of WPDP, and CPDP-ALL produce very bad defect prediction results in terms of G-measure. The finding that learning from all cross training data without data filtering will lead to higher PF is consist with what Turhan et al. found [18].

Results of this simulation experiment provide empirical evidence showing that learning defect prediction models from open-source project data is a feasible way for proprietary projects lacking local historical data. However simply learning prediction models from all available open source project training data without data selection is not a good practice.

Data Selection for Cross Project Defect Prediction

In Section 3.7.2 we show that learning from open-source projects is theoretically possible when using the right training data. Consequently, the problem is how to select these appropriate cross training data. In this study we proposed a new training data selection method based on data similarity. Our goal is to learn from training data that have similar distribution with that of the test set. Our proposed method achieves this goal by two operations: instance selection and feature subset selection.

Considering the various development environments and inherent characteristics of different projects, it is possible that the historical data extracted from different projects are different, i.e., the distribution of the data may be different from each other from a mathematical point-of-view. Thus an intuitive idea for cross project defect prediction is to learn from historical projects that are similar to the target project. The KNN filter introduced by Turhan et al. is a typical way to find "similar" training data for model training [18]. However, our proposed method do instance selection in a different way, we select training instances on the granularity of data set, i.e., we select or filter out some cross training sets rather than some individual data instances. We argue that the properties of each training/test set are reflections of the properties of the project that generated that data set (or in other words, the "context" of the project). Also it is possible to find "similar" projects to learn from by measuring similarity between data sets. In the proposed data selection method, we adopt a simple and intuitive strategy for instance selection: for each test set, we only learn from the top N closest training sets.

After instance selection, we further employ feature subset selection to filter out features that cause differences between distributions of training data and test data (i.e., unstable features). The intuition is that we should learn on these "common" feature subsets in case training data and test data are different. However, we have to point out that feature subset selection is a trade-off between data similarity and training information. The more unstable features we filter out, the more similar the training data and test data will be, while at the same time the more supervised information in

the training data for model learning is lost. In our proposed training data selection method, we adopt a data change analysis framework to identify these unstable features.

It is natural that the common feature subset between different training set and test set pairs are different. According to our experimental setting, we try to learn defect prediction models for proprietary projects from multiple open-source projects. So to combine the knowledge of multiple training sets, we adopt the bagging ensemble method to get the final prediction results for the test set. More specifically, we first learn prediction models on each individual training set after feature subset selection, and then apply these prediction models to the test set and use majority voting to combine prediction results.

Figure 3.3 gives an overview of our proposed training data selection method for cross project defect prediction. We then describe technical details about measuring similarity between data sets, identifying unstable features, the bagging ensemble strategy, and empirical evaluation of the proposed method, respectively.

Input: training sets $D_{train}^{(1)}$, $D_{train}^{(2)}$, ...,$D_{train}^{(m)}$;
test set D_{test};
top_N; {learning from top_N closest training sets}
FSS_ratio; {the ratio of unstable features filtered out}
score_thre; {the threshold for bagging.}
Output: Prediction results for D_{test}.
Procedure:

1. For each training set $D_{train}^{(i)}$ (i = 1, 2, ..., m), calculate the distance between $D_{train}^{(i)}$ and D_{test};
2. Select top_N training sets with lowest distance to the test set;
3. For each selected training set of step 2, remove FSS_ratio $\times$ 100 percent of features that are most unstable;
4. Learn prediction models on each selected training set (step 2) after feature subset selection (step 3), and ensemble prediction results from each prediction model by using bagging (i.e, majority voting with a threshold score_thre).

Figure 3.3: Overview of the proposed training data selection method

Measuring Data Similarity: The data similarity measuring method we adopted is derived from

the data change detection and analysis framework proposed by Hido et al. [87]. Given a training set D_{train} that comprises of M labeled instances $\{(x^{(1)}_{train}, y^{(1)}_{train}), (x^{(2)}_{train}, y^{(2)}_{train}), ...(x^{(M)}_{train}, y^{(M)}_{train})\}$, and a test set D_{test} that comprises of N unlabeled instances $\{(x^{(1)}_{test}), (x^{(2)}_{test}), ...(x^{(N)}_{test}\}$, the data similarity measuring problem is to calculate the similarity between the marginal distribution of the training set (noted as $P(X_{train})$) and that of the test set (noted as $P(X_{test})$). The key idea of the data similarity measuring solution is as follows:

1. Randomly sample K instances from training set D_{train}, noted as SAM_{train}, and K instances from test set D_{test}, noted as SAM_{test}. Then attach a hypothetical label 0 to each instance of SAM_{train}, and 1 to each instance of SAM_{test}.

2. Combine SAM_{train} and SAM_{test} into a new data set SAM whose size should be $2K$. Train a classifier C on SAM in a supervised fashion and evaluate its accuracy on SAM.

3. Use the accuracy of the classifier in Step 2 as a measurement of distance between D_{train} and D_{test}.

Figure 3.4 shows an overview of the data similarity measuring method, where $\triangle$ and $\square$ indicate instances of SAM_{train} and SAM_{test}, respectively. If marginal distributions of D_{train} and D_{test} are actually different, instances of SAM_{train} and SAM_{test} should be correctly classified by the classifier. Thus a high classification accuracy indicates a difference between D_{train} and D_{test}. For example, if $P(X_{train}) = P(X_{test})$, the classification accuracy will be about 0.5. However, if the accuracy is significantly larger than 0.5, we can infer that $P(X_{train})$ and $P(X_{test})$ are different with each other.

In this study, we employ the Logistic Regression algorithm to build the classifier mentioned in Step 2 above. We evaluate the classifier using 5-folds cross validation. To eliminate the potential bias caused by the data sampling operation in Step 1, we repeat Step 1 and Step 2 for 10 times and use the mean value of accuracies from these 10 rounds as the final accuracy (noted as Acc). Our calculation to convert classification accuracy to data distance is as follows:

$$Distance(D_{train}, D_{test}) = 2 * (Acc - 0.5)$$

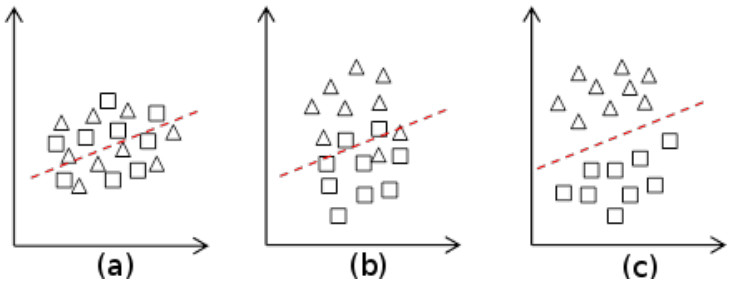


Figure 3.4: Overview of the data similarity measuring method, these pictures show examples that distribution of two data sets are (a) very close to each other, (b) partly close, and (c) totally different, respectively.

Identifying Unstable Features: To identify which part of features cause the differences between D_{train} and D_{test}, we revisit the hypothetical data set *SAM* generated for measuring data distance. The intuition is that features that play a dominant role in the classification are the ones that characterize the difference. Hido et al. suggested to identify unstable features by exploring the structure of classifier C learned from *SAM* [87]. For example, if C was constructed using the Logistic Regression algorithm, features with high regression coefficients would be more unstable; or if C was constructed using the Decision Tree algorithm, features close to the root of the tree can be seen as unstable ones [87].

In this study, we identify unstable features by comparing the Information Gain associated with each feature. More specifically, we calculate information gain of each feature against data set *SAM*, and see features with the highest information gains as unstable features.

Ensemble Prediction Results: The intuition of adopting bagging to merge prediction results produced by each individual defect prediction model is that the stable feature subset selected may be different on different training sets, thus we need an ensemble mechanism to combine prediction results derived from heterogeneous training sets. Bagging is essentially a method to eliminate

the potential prediction bias caused by each individual predictor by integrating their predictive power [88]. We employed bagging to combine defect prediction results generated by different training sets after feature subset selection. The process of the bagging ensemble method can be described as follows:

Given a set of classifiers $\{C_1, C_2, ...C_p\}$ and the test set D_{test}, for each instance of D_{test}:

1. Get prediction result of each classifier for that instance, noted as r_i (i = 1, 2, ...p).

2. Compute the voting score of that instance as:

$$score = \sum_{i=1}^{p} score_i$$

   where $score_i = 1$ if r_i says the prediction result is "defect-prone", otherwise $score_i = 0$.

3. If the voting score is greater than a pre-defined threshold *score_thre*, the final prediction result of that instance will be set as "defect-prone".

3.7.3 Evaluation

To evaluate the effectiveness of our proposed training data selection approach, we compare its performance with that of the KNN filter [18]. During the comparisons, we set the parameter K=10 for the KNN filter. Also for our data selection approach, we set *top_N=10* and *score_thre=0.5* in all experiments. As for the parameter *FSS_ratio* for feature subset selection, we try *FSS_ratio* from 0.1 to 0.9 since it is a trade-off between data similarity and training information, and we observe the change of prediction performance.

Figure 3.5 illustrates the change of prediction performance when using different parameter settings for feature subset selection, where the X-axis indicates how many unstable features are removed. We can see that the Random Forest, Naïve Bayes and Logistic Regression learners can achieve highest performance after removing 60%, 70%, and 80% of features, respectively. An

intuitive conclusion is that cross project defect prediction can achieve best performance by only learning from 20% to 40% features. What's more, we can see that the Random Forest learner performs worse than Naïve Bayes and Logistic Regression, while in the within project learning scenario Random Forest performs best. A potential explanation for this observation is that there may be over-fitting happening for Random Forest. Our suggestion is to use simple learners such as Naïve Bayes rather than complicated learners such as Random Forest when doing cross project defect prediction.

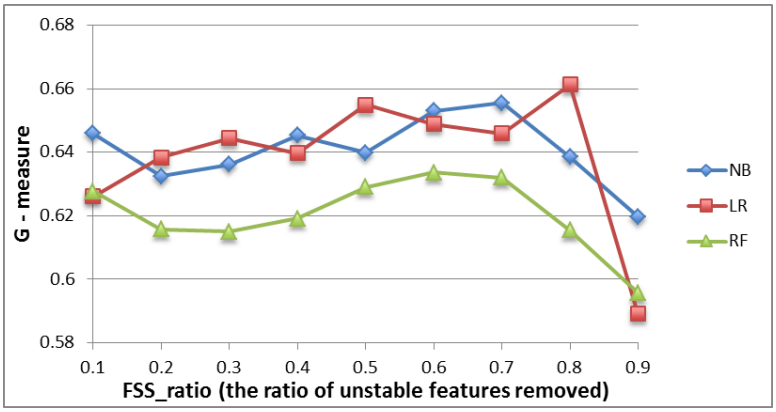
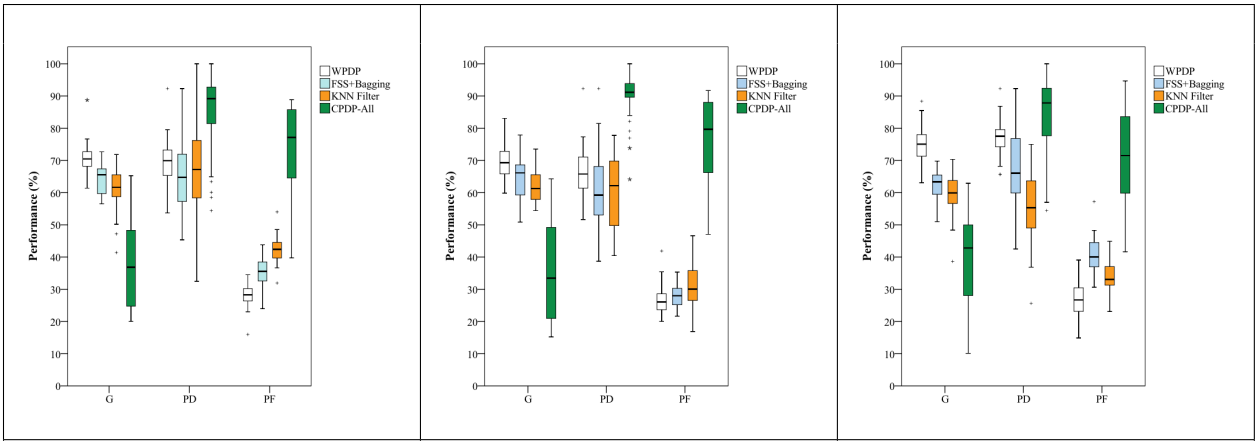


Figure 3.5: Comparison of G-measure under different parameter setting for feature subset selection.

We then compare the best performance our proposed method with those of the KNN filter and within project learning (WPDP). Figure 3.6 illustrates the comparison details. For all three learning algorithms we tested, our proposed method produces relatively higher G-measure values than the KNN filter. For prediction models using Naïve Bayes and Logistic Regression learner, our method produces lower PD as well as lower PF, and for prediction models using Random Forest, our method produces both higher PD and PF. An empirical conclusion observed in cross project defect prediction using static code metrics is that cross training data may generate prediction results with comparable or even higher PD at the cost of a higher PF [18]. A key task of cross project defect prediction is to find methods that can reduce the PF. Figure 3.6 shows that our proposed methods can largely reduce PF while it doesn't cause big damage to PD if using Naïve Bayes or



(a) Naïve Bayes (b) Logistic Regression (c) Random Forest

Figure 3.6: Comparison of performance between our proposed method and the KNN filter, where WPDP presents the within project defect prediction, FSS+Bagging presents cross project defect prediction using our proposed data selection method, and KNN Filter presents cross project defect prediction using the KNN filter.

Logistic Regression. As for Random Forest, again, we do not recommend it for cross project defect prediction because of its bad performance.

To further support our observation from Figure 3.6, we employed Mann-Whitney U test to compare the performance of KNN filter and our method. Table 3.3 gives an overview of the significance test results, where "Win" indicates that performance of our method is better than KNN filter, "Loss" means that KNN filter is better, and "Tie" means no significant difference observed between these two methods. The significance test results show that our proposed method performs better or equally with the KNN filter for most test sets.

Table 3.3: Comparision of Prediction Performance Between Our Proposed Method and the KNN Filter. This Comparision Result is Based on Significance Test Results Using Mann-Whitney U Test at Confidence Level 0.95.

Learner	#Win	#Tie	#Loss
NaïveBayes	22	8	4
Logistic Regression	15	12	7
Random Forest	17	12	5

Compared with the KNN filter, our proposed method can generate better results for cross

project defect prediction with regards to the G-measure. Also for two out of the three learners tested, our method can largely reduce the PF without causing big damage to PD.

Necessity of Cross Project Defect Prediction

Some studies argue that learning from other projects is a serious problem while some other argue that the performance of cross project defect prediction is comparable with that of within project defect prediction [18–20]. Our findings in this study are consistent with what Turhan et al. found and support the conclusion that learning from other projects without data filtering will cause very high PF [18]. Also, considering the expensive cost of project measurement and data maintenance program [89], and the fact that local historical data are not always available (e.g., the first release of a new project), we argue that cross project defect prediction is necessary.

Additionally, data filtering is needed because the characteristics of the training data and the test data may be significantly different. For example, for the open-source projects and proprietary projects we investigated, there is a very intuitive difference: the ratio of defect-prone classes of proprietary projects are much lower than that of open-source projects.

3.8 Summary

These results lead to the following question. Can proprietary data predict for open source data? Zimmermann et al. have shown us that it was not possible with Internet Explorer and Firefox. However, in later chapters, our experiments with LACE indicate that this is possible. In subsequent chapters we explain the design and operation or LACE (Chapter 4) and evaluate its performance for both privacy (IPR) and utility (CPDP) (shown in Chapter 5 and Chapter 6).

From this chapter we also see that cross project defect prediction is challenging but can be improved through the use of transfer learning techniques. This result, particularly for the open-

source and proprietary study, intensifies the need for research into privacy issues associated with the sharing component of cross project defect prediction. Without this privacy research, cross project defect prediction will be limited to studies with only open source data.

Chapter 4

LACE Design and Operation

4.1 Introduction

The ability to build quality defect predictors for proprietary data using open source data as shown in the previous chapter, highlights the increasing importance for privacy research in software defect prediction. Since confidentiality is so important, and since standard privacy methods damage data mining efficacy, we have been engaged for some time on research to build better privacy algorithms for software defect prediction. Our previous work [11, 12] allowed data owners to share a minimized and obfuscated version of their data on an individual level. In this dissertation we repeat those results with a more comprehensive parameter tuning experiment (Chapter 5) and we extend that work with *private multiparty data sharing* (Chapter 6) based on the following scenario.

Consider the problem of l parties (data owners) $P_0 ... P_{l-1}$, each with local data, x_i. They want to securely work together to create a *private cache* containing pooled minimized and obfuscated data from all parties involved. Each data owner P_i determines what data to add to the private cache based on what others have added previously. The final private cache can then be used by others for cross project defect prediction. According to Lindell et al. [41], this problem is a special case in cryptography where a set of parties with private data wishes to jointly compute some function of

their data.

Given that our aim is to share data for the purpose of cross project defect prediction, we include this second mode of *LACE* where multiple data owners can team up and share data guided by a novel multiparty privacy protocol. Figure 4.1 illustrates how LACE works for private multiparty data sharing with three data owners.

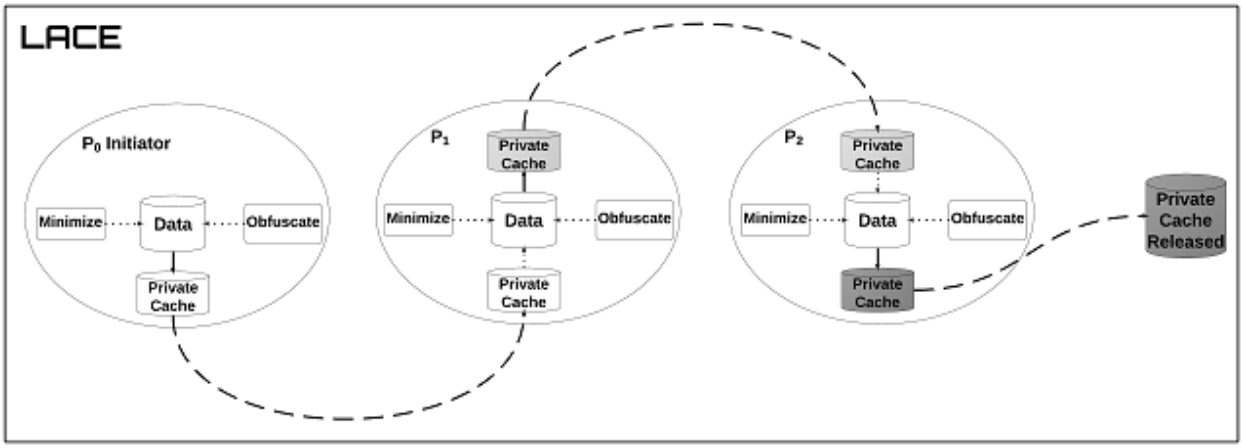


Figure 4.1: Example of three data owners teaming up in LACE (the Large-scale Assurance of Confidentiality Environment) to produce a private cache for cross project defect prediction.

First LACE randomly selects a data owner P_0 as the initiator of the privacy protocol. P_0 then minimizes and obfuscates its data then adds these to the *private cache*. The cache is then randomly passed to another data owner P_1, who minimizes it's data based on the data in the private cache then obfuscates it and passes the result to the private cache. The process at P_1 is repeated at the randomly chosen P_2 data owner, who then releases the final private cache as a training set for cross project defect prediction.

Minimization in LACE takes place in two steps:

1. Instance selection with CLIFF, which is an instance selector that returns instances that best predict for the target class;

2. Instance selection via incremental learning with the leader-follower algorithm (LeaF) [90] which is an online technique for clustering data. It is this algorithm that facilitates collabo-

ration among data owners where they submit data to the cache that are *dissimilar* to what's already there.

Obfuscation in LACE is handled by applying our MORPH algorithm on the minimized data. MORPH reflects on the *boundary* between this instance and the nearest instance of another class (and MORPH's *restricted mutation* policy never pushes an instance across that boundary).

Note the key features of this protocol:

- Private data remains with the data owner behind the firewalls.
- All the algorithms are run behind firewalls by data owner. Hence, in LACE, there is no need for a central server or some third party privatization service.
- Most data is never shared. LACE's instance selection prunes away most of the data (which means that the pruned data is completely private).
- The shared data is privatized such that queries to that data return very different values than in the raw data. In other words, we reduce the risk is sensitive attribute disclosure.
- MORPH's obfuscation of the data does not change the classifications of the training data. That is, this protocol obfuscates data without damaging data mining efficacy.

In addition to the privacy algorithms, LACE also includes *IPR (increased privacy ratio)* to measure the privacy threat of sensitive attribute disclosure for our LACEd data. We expand on the basics of LACE, the algorithms and IPR in the following sections.

4.2 LACE

We designed LACE based on the success and failures of previous work on multiparty computation. As explained by Vaidya et al. [91], the advantage of perfectly secure multiparty computation is that nothing is revealed. They offer a simple example of such a computation. Suppose we want to know

the average age of everyone attending a conference. First, we generate a large random number R and pass it to a random attendee. The attendee adds their age and passes the sum to another attendee (selected at random). This repeats till all attendees have been sampled at which point the sum returns to the origin. After subtracting R, we have the sum of the ages from which we can compute the average sum.

The benefits of this protocol are that, if R is kept private, then no single participant can "decode" the passed value to find the sum of the ages. Also, if the ordering of the sampled attendees is also kept private, then no pair of attendees a, c can compare their numbers to determine the age of the attendee b who was sampled between a and c.

Vaidya et al. discuss an experiment with a distributed data miner (based on C4.5) that used a variant of the above multiparty computation, every time it searched data from different organizations. They declared that experiment a failure, for two reasons. Firstly, the network overhead of that approach was prohibitive. Secondly, this approach conducted so many queries across different sites that it was possible for pairs of sites to collude to "decode" the passed values.

Our analysis of the failed Vaidya et al. experiment was that it is a mistake to conduct multiple *micro-queries* of a distributed data sources. Rather, what is more practical, is a one-pass *coarse-grained query* over a random sample of those source. Such a single-pass random sampling approach mitigates against collusion. Further, it reduces the network traffic associated with the query.

LACE is such a single-pass randomized query. Instead of a total sum of numbers, its desired outcome is a private cache containing exemplars from each site. LACE proceeds as follows:

1. Each data owner applies CLIFF to their data. Only the data selected by CLIFF are used in LACE.

2. The initiator is chosen at random and applies LeaF (Section 4.2.4) to minimize its CLIFFed data.

3. The results are then obfuscated with MORPH (Section 4.2.2), and are the beginnings of the private cache.

4. The private cache is sent to the second randomly chosen site (data owner), where they test each instance of their CLIFFed data using LeaF and the private cache. The test involves each instance finding its nearest exemplar in the private cache and if their class labels are different and/or they are a certain distance away then that instance is MORPHed and added to the cache.

5. Once this is complete the private cache moves on to the next random data owner and the process repeats.

6. The protocol is complete when all data owners involved have had a chance to contribute to the private cache.

When implementing LACE, we make the following assumptions:

- There must be at least three data owners involved in the process. This is because for only two parties, the result will reveal the input of the other party, thus eliminating privacy [91].

- Since data is pooled into a private cache for defect prediction, each party must provide data with the same features or attributes.

- Data involved in LACE are *not extreme*. For example in a case where a bunch of startups and Microsoft contributed to a private cache. Even with the random perturbation in MORPH, it will be obvious which defect data came from Microsoft Windows vs. all of the others because Windows will have LOC orders of magnitude greater than anyone else.

4.2.1 Minimization with CLIFF

One core aspect of LACE is to look at less data. In this section we describe CLIFF which is used to minimize data.

Initially, CLIFF was constructed to achieve the goal of reducing the drawbacks of the k-Nearest Neighbor (kNN) algorithms by Hart [92]. Since their invention, many researchers have explored intelligent *instance selection methods* to address the drawbacks with k-Nearest Neighbor algorithms [40, 93–114]. These drawbacks include, (a) the high computation costs caused by the need for each test sample to find the distance between it and each training sample (b) the storage required to hold the entire dataset in memory; (c) the effects of outliers which can negatively affect the accuracy of the classifier; (d) the negative effect of data sets with non-separable and/or overlapping classes; and (e) the effects on noise on kNN inference [39, 40]. Noise can lead to *brittleness* i.e. when random changes to an instance leads to a major change in the inferences drawn from those instances.

A standard solution to these drawbacks is some *instance selection* algorithm that replaces all the data with X% fewer instances. Instance selection can mitigate the effects of noise and therefore reduce brittleness.

Research in instance selection is an active field of study [95, 98, 102, 112, 115–119]. Instance selection involves selecting a subset of instances from the original training set. Using what Dasarathy [95] terms as *edit rules*.

CLIFF is our solution for reducing or eliminating these drawbacks. The success of CLIFF when compared with other instance selectors allowed us to see that we could maintain the utility of data even with a minimized data set. CLIFF assumes tables of training data can be divided into *classes*. For example, for a table of defect data containing code metrics, different rows might be labeled accordingly (defective or not defective).

CLIFF executes as follows:

- For each column of data, find the *power* of each attribute sub-range; i.e., how much more frequently that sub-range appears in one class more than any other.
- In prior work [120], at this point we select the sub-range with the highest *power* and removed

all instances without this sub-range. From the remaining instances, those with sub-ranges containing the second highest *power* are kept while the others are removed. This process continued until at least two instances were left or to the point before there were zero instances left. In this work, to control the amount of instances left by CLIFF, we find the product of the *powers* for each row then,

- Remove the less *powerful* rows.

The result is a reduced data set with fewer rows. In theory, this reduced data set is less susceptible to privacy threats such as sensitive attribute disclosure discussed in Chapter 2.

Algorithm 1: *Power* is based on BORE [121]. First we assume that the target class is divided into one class as *first* and the other classes as *rest*. This makes it easy to find the attribute values which have a high probability of belonging to the current *first* class using Bayes theorem. The theorem uses evidence E and a prior probability $P(H)$ for hypothesis $H \in \{first, rest\}$, to calculate a likelihood (hereafter, *like*) of the evidence selecting for one class:

$$like(H|E) = P(E|H) \times P(H).$$

This calculation is then normalized to create probabilities:

$$P(first|E) = \frac{like(first|E)}{like(first|E) + like(rest|E)} \tag{4.1}$$

Jalali et al. [121] found that Equation 1 was a poor ranking heuristic for low frequency evidence. To alleviate this problem the support measure was introduced. Note that *like(first|E)* is also a measure of support since it is maximal when a value occurs all the time in every example of one class. Hence, adding the support term is just:

$$P(first|E) * support(first|E) = \frac{like(first|E)^2}{like(first|E) + like(rest|E)} \tag{4.2}$$

To compute the *power* of a sub-range, we first apply *equal frequency binning* to each attribute

in the data set. Equal frequency binning divides the range of possible values into n bins or sub-ranges, each of which holds the same number of attribute values. However, to avoid duplicate values being placed into different bins, boundaries of every pair of neighboring bins are adjusted so that duplicate values should belong to one bin only [122]. For these experiments, we did not optimize the value for n for each data set. We simply used *n=10* bins. In future work we will dynamically set the value of n for a given data set.

Algorithm 1 Finding sub-range *Power*.

```
1: Power(D, E) {D is the data-set, and E is a set of sub-ranges for a given attribute}
2: Partition(D) ↦ C {Returns data partitioned according to the class label.}
3: PR ← ∅ {Initialize sub-ranges with power for each sub-range in E}
4: for j=0 to # of class values in |C| do
5:    first ← C_j
6:    rest ← C_≠j
7:    p_first ← |first|/|D| {Probability of first data}
8:    p_rest ← |rest|/|D| {Probability of rest data}
9:    for k=0 to # of sub-ranges in |E| do
10:      like(first|E_k) ← number of times E_k appears in first × p_first
11:      like(rest|E_k) ← number of times E_k appears in rest × p_rest
12:      power_k ← like(first|E_k)^2 / (like(first|E_k)+like(rest|E_k))
13:      PR ← power_k
14:   end for
15: end for
16: return PR
```

Next, *CLIFF* selects $p\%$ of the rows in a data-set D containing the most powerful sub-ranges. The matrix M holds the result of *Power* for each attribute, for each class in D. This is used to help select the rows from D to produce *D'* within *CliffSelection*.

The *for*-loop in Lines 3 to 9 of Algorithm 2 iterates through attributes in D and *UniqueRanges* is called to find and return the unique sub-ranges for each attribute. Inside that loop, at Lines 5 to 8, a nested *for*-loop iterates through the unique sub-ranges for a given attribute and *Power* is called to find the power of each sub-range. Last, once the *power*s are found for each attribute sub-range, *CliffSelection* is called to determine which rows in D will make up the final sample in *D'*:

- Partition D by the class label;

- For each row in each partition, find the product of the *power* of the sub-ranges in that row;
- For each partition, return the p percent of the partitioned data with the highest *power*.

An example of how CLIFF is applied to a data set is described in Section 4.2.3.

Algorithm 2 The *CLIFF* algorithm.

```
1: CLIFF(D, p) {D is the original data-set, and p is the percentage of data to be returned}
2: M ← ∅ {Initialize sub-range power for each attribute}
3: for i=0 to # of attributes in D do
4:    UniqueRanges(D_i) ↦ R_i {Returns set of unique sub-ranges for a given attribute}
5:    for j=0 to # of sub-ranges in R_i do
6:       Power(D, R_i) ↦ PR_j {Returns the sub-ranges with their powers for each class}
7:       M ← PR_j
8:    end for
9: end for
10: CliffSelection(D, p, M) ↦ D' {Returns p of the original data}
11: return D'
```

4.2.2 Obfuscation with MORPH

MORPH is an instance mutator that changes the numeric attribute values of each row by replacing these original values with *MORPHed* values. MORPH takes care never to change an instance such that it moves across the boundary between the original instance and instances of another class.

The MORPHed instances are created by applying Equation 4.3 to each attribute value of the instance.

$$y = x \pm (x - z) * r \tag{4.3}$$

Let $x \in D$ be the original row to be changed, y the resulting MORPHed row and $z \in D$ the Nearest Unlike Neighbor (NUN) of x. NUN is the nearest neighbor of x whose class label is different from x's class label (distance is calculated using the *Euclidean* distance). The random number r is calculated with the property:

$$\alpha \leq r \leq \beta.$$

In previous work we used $\alpha = 0.15$ and $\beta = 0.35$. These values for α and β were chosen in an effort to keep the MORPHed value close enough to the original in order to maintain the utility of the data set but far enough to keep the original data private. For the experiments in Chapter 5 of this dissertation we expand this range of values to $\alpha = 0$ and $\beta = 1$.

A simple hashing scheme lets us check if the new instance *y* is the same as an existing instance (and we keep MORPHing *x* until it does not hash to the same value as an existing instance). Hence, we can assert that none of the original instances are found in the final data-set.

An example of how MORPH is applied to a data set is described in Section 4.2.3.

4.2.3 Illustrative Example of CLIFF&MORPH

Let us consider the abbreviated version of the *ant-1.3* data set shown in Table 4.1a. Definitions for these attributes as well as other attributes of data used in the experiments in this dissertation are shown in Table 2.1. This data set contains ten attributes. One dependent attribute (class) and nine independent attributes. Each row is labeled as 1 (containing at least one defect) or 0, (having no defects). The first column holds the row number and each cell contains the original metric values.

The result of applying CLIFF&MORPH is shown in Table 4.1e. To get to that point, first the original data is binned using equal frequency binning because CLIFF is designed to handle discrete data. The result of this is shown in Table 4.1b. For example the attribute values of *wmc* are replaced by two sub-ranges of values ([3-6] and (6-14]). Here all values from 3 to 6 inclusive are placed in the first sub-range and all values between 6 and 14 (not including 6) are placed in the last sub-range.

Following this, each sub-range is the ranked according to Equation 4.2. To find the *power* of each sub-range, we first divide the data into *first* and *rest*. For this example, let us say that all the rows with the 0 class label are *first* while the others are *rest*. Figure 4.2, shows an example of finding the *power* of (6-14] for attribute *wmc* and Table 4.1c shows the *power* values for all the sub-ranges of Table 4.1b.

Table 4.1: Example of CLIFF&MORPH. (a) Shows the original data and is an abbreviated version of *ant-1.3*. (b) Data from *a* binned using equal frequency binning. (c) Power values for each sub-range in *b*. (d) CLIFF result. (e) MORPH result.

(a) Partial ant-1.3 Defect Data

#	wmc	dit	noc	cbo	rfc	lcom	ca	ce	loc	class
1	11	4	2	14	42	29	2	12	395	0
2	14	1	1	8	32	49	4	4	297	1
3	3	2	0	1	9	0	0	1	58	0
4	12	3	0	12	37	32	0	12	310	0
5	6	3	0	4	21	1	0	4	136	0
6	5	1	5	12	11	8	11	1	59	0
7	4	2	0	3	16	0	0	3	59	0
8	14	1	0	24	63	63	20	20	822	1

(b) ant-1.3 After Equal Frequency Binning

#	wmc	dit	noc	cbo	rfc	lcom	ca	ce	loc	class
1	(6-14]	[1-4]	[0-5]	(8-24]	(21-63]	(8-63]	(2-20]	(4-20]	(136-822]	0
2	(6-14]	[1-4]	[0-5]	[1-8]	(21-63]	(8-63]	(2-20]	[1-4]	(136-822]	1
3	[3-6]	[1-4]	[0-5]	[1-8]	[9-21]	[0-8]	0	[1-4]	[58-136]	0
4	(6-14]	[1-4]	[0-5]	(8-24]	(21-63]	(8-63]	0	(4-20]	(136-822]	0
5	[3-6]	[1-4]	[0-5]	[1-8]	[9-21]	[0-8]	0	[1-4]	[58-136]	0
6	[3-6]	[1-4]	[0-5]	(8-24]	[9-21]	[0-8]	(2-20]	[1-4]	[58-136]	0
7	[3-6]	[1-4]	[0-5]	[1-8]	[9-21]	[0-8]	0	[1-4]	[58-136]	0
8	(6-14]	[1-4]	[0-5]	(8-24]	(21-63]	(8-63]	(2-20]	(4-20]	(136-822]	1

(c) ant-1.3 Power Values

#	wmc	dit	noc	cbo	rfc	lcom	ca	ce	loc	class
1	0.13	0.56	0.60	0.28	0.13	0.13	0.28	0.17	0.28	0
2	0.13	0.06	0.06	0.03	0.13	0.13	0.03	0.03	0.03	1
3	**0.50**	**0.56**	**0.56**	**0.28**	**0.50**	**0.50**	**0.75**	**0.40**	**0.50**	**0**
4	0.13	0.56	0.56	0.28	0.13	0.13	0.75	0.17	0.28	0
5	0.50	0.56	0.56	0.28	0.50	0.50	0.75	0.40	0.50	0
6	0.50	0.56	0.56	0.28	0.50	0.50	0.28	0.40	0.50	0
7	0.50	0.56	0.56	0.28	0.50	0.50	0.75	0.40	0.50	0
8	**0.13**	**0.06**	**0.06**	**0.03**	**0.13**	**0.13**	**0.03**	**0.04**	**0.03**	**1**

(d) CLIFF Result

#	wmc	dit	noc	cbo	rfc	lcom	ca	ce	loc	class
3	3	2	0	1	9	0	0	1	58	0
8	14	1	0	24	63	63	20	20	822	1

(e) MORPH Result

#	wmc	dit	noc	cbo	rfc	lcom	ca	ce	loc	class
3	2	2	0	1	5	5	2	0	1	0
8	15	1	0	26	67	68	22	21	879	1

Next, the *power* of each row is calculated by finding the product of the sub-range *powers* of each row. In this example the row with the highest *power* for each class is selected. In this case that is row 3 for the 0 class label and row 8 for the 1 class label. This result is shown in Table 4.1d.

Finally, we apply MORPH this result according to Equation 4.3 to obtain the result in Table 4.1e.

$$
\begin{array}{lcl}
\text{E} & = & \text{(6 - 14]} \\
\text{P(first)} & = & {}^{6}/_{8} \\
\text{P(rest)} & = & {}^{2}/_{8} \\
\text{freq(E|first)} & = & {}^{2}/_{6} \\
\text{freq(E|rest)} & = & {}^{2}/_{2} \\
\text{like(first|E)} & = & {}^{2}/_{6} \times {}^{6}/_{8} = 0.25 \\
\text{like(rest|E)} & = & {}^{2}/_{2} \times {}^{2}/_{8} = 0.25 \\
\text{P(first|E)} \times \text{support(first|E)} & = & \frac{0.25^2}{0.25+0.25} = 0.13
\end{array}
$$

Figure 4.2: Finding the *power* of $(6-14]$

4.2.4 LeaF: Leader Follower Algorithm

LeaF is a leader-follower algorithm [90]. It is an online incremental technique for clustering data. The cluster centers are the "leaders" and all other instances are the "followers". For this work we are only interested in the leaders. The basic algorithm works as follows: (1) initialize cluster centers, then (2) for each instance in the data, find its nearest center. If the distance to the center is less than a user defined distance, then (3) update the cluster. Otherwise, create a new cluster with the instance as the center.

In the scenario of collaboration among multiple data owners, LeaF is applied to each instance selected by CLIFF from the party's data set to determine if it should be included in the private cache. To fit our work, we adapt LeaF in four ways:

1. The cluster centers are never updated to create centroids.

2. The initial centers are the two instances I_1, I_2 that are farthest from each other in the data. I_1 and I_2 are found using the *choose distant object* algorithm by Faloutsos et al. [123]. We call the distance between I_i, I_2 the *separation* of the data.

3. We use the value $d = separation/10$ to determine if data from a data owner should be included in the private cache (new data is added to the cache if it falls more than d away from its nearest exemplar and their class attribute values are different.) [1]

4. Prior to a new instance being added to the cache, it is first MORPHed using the method described above. The source code for LeaF is shown in Listing 1 and the source code for the *private cache* is shown in Listing 2. LeaF is only used by the data owner who is the initiator for multiparty data sharing. All other data owners that follow, use the *private cache*. They input their CLIFFed data and the current private cache. The result is an updated cache whose new MORPHed exemplars adhere to the conditions for entering the private cache.

4.3 How are privatized data candidates evaluated?

In our work, we seek to defend data against the privacy threat of sensitive attribute disclosure. To measure if LACEed data reduces this threat we use *IPR* which models a sensitive attribute disclosure attack using *background knowledge*.

4.3.1 IPR: Increased Privacy Ratio

For IPR, we assume the role of an attacker armed with some background knowledge from the original data set and also supplied with the private data set. When the attacker predicts the sensitive attribute value of a target we use the original data to see if the prediction is correct. If it is we

[1] Note that the smaller the distance value the more data gets added, the larger the value the less data gets added to the private cache. We leave it to future work to determine what would be the best distance value to use.

Listing 1 Source code for LeaF main function.

```
(defn leaf
 "Returns matrix of exemplars from data and maybe the distance value, dist-s.
  Arguments:
  data matrix

  Options:
  :p (default 0.1)
  used to calculate dist-s=separation/10
  :only-exemplars (default true)
  return only the exemplars otherwise return exemplars vectored with dist-s.
  :distance-fnc (default euclidean-distance)
  returns euclidean distance between two points."
 [data & {:keys [p only-exemplars distance-fnc]
          :or {p 0.1 only-exemplars true distance-fnc euclidean-distance}}]
 (let [pivots  (choose-distant-objects ; Find two farthest points, east and west.
                data :distance-fnc distance-fnc)
       east    (nth data (first pivots))
       west    (nth data (second pivots))
       dist-ew (distance-fnc    ; Distance between east and west
                 (butlast east)
                 (butlast west))
       dist-s  (* p dist-ew)]   ; Divide distance by 10.
   (loop [dat data result [east west]]
     (if (empty? dat)
       (if only-exemplars                        ; If only-exemplars is true,
         (matrix (remove #(= nil %) result))     ; return only the exemplars,
         [(remove #(= nil %) result) dist-s])    ; otherwise return exemplars
       (recur                                    ; and distance, dist-s.
         (rest dat)
         (conj result (let [ans (get-nearest
                                  (first dat)
                                  (remove #(= nil %) result)
                                  :distance-fnc distance-fnc)
                            pt  (first ans) ; Nearest exemplar from result.
                            dist-pt (second ans)
                            label (last pt)]
                        (if (and  ; If instance class label = nearest exemplar label
                              (= (last (first dat)) label)  ; and distance to nearest
                              (< dist-pt dist-s))           ; is less than dist-s
                          nil                               ; return nil
                          (first dat)))))))))               ; else return instance.
```

consider this as sensitive attribute disclosure otherwise it is not. Listing 3 shows the source code for sensitive attribute disclosure.

Recall that we assume that the attacker has access to a privatized version (T') of an original data

Listing 2 Source code for the private cache.

```
(defn private-cache
 "Returns updated private cache, with MORPHed exemplars.
  Arguments:
  X, the current cache
  cliff-data, new CLIFFed data
  dist-s, distance boundary from LeaF
  r, boundary for MORPHing to nearest unlike neighbor

  Options:
  :sav (default [10 20])
  vector containing the sensitive attribute value(s) and the class attribute is last.
  :distance-fnc (default euclidean-distance)
  returns euclidean distance between two points."
 [X cliff-data dist-s r & {:keys [sav distance-fnc]
                           :or {sav [10 20] distance-fnc euclidean-distance}}]
 (loop [dat cliff-data exemplars X]
   (if (empty? dat)
     (let [cache (remove #(= nil %) exemplars)]
       cache)
     (recur
       (rest dat)
       (conj
         exemplars
         (morph-one-exemplar  ; Returns MORPHed exemplar if it meets conditions,
           (first dat)        ; otherwise returns nil.
           dist-s r))))))

```

set (T), and knowledge of non-sensitive QID values for a specific target in T. We refer to the latter as a query. For our experiments we randomly generate up to 1000 of these queries, $|Q| \leq 1000$ (Section 4.3.3 describes how queries are generated).

For each query, q in a set $Q = \{q_1, ..., q_{|Q|}\}$, G_i^* is a group of rows from any data set which matches q_i. Hence, let G_i be the group from the original data set and G_i' be the group from the privatized data set which matches q_i. Next, for every sensitive attribute sub-range in the set $S = \{s_1, ..., s_{|S|}\}$, we denote the idea of the most common sensitive attribute value (otherwise known as the attacker's best guess) as $s_{max}(G_i^*)$. Listing 3 shows the source code for finding this common sensitive attribute value. The functions takes as input G_i' (data-q), the privatized data set (all-data) and a list of sensitive attributes. For each sensitive attribute a *best guess* sensitive value is returned.

However is the G_i' is empty, a question mark is returned instead.

Listing 3 Source code for the attacker's best guess (sad++) main function.

```
(defn sad++
 "Returns the sensitive attribute disclosure of one or more sensitive attributes.
  Arguments:
  data-q: Instances resulting from a query.
  atts: Vector of sensitive attribute values."
 [data-q atts]
 (loop [att atts result []] ; For each senstive attribute in att
   (if (empty? att)
     result
     (recur
       (rest att)
       (conj result
             (cond
               (empty? data-q) "?" ; If data-q empty, return ?.

               (= (nrow data-q) 1) ; If data-q has one instance,
               (nth (first data-q) (first att)) ; return senstitive value.

               :else (first (first        ; If data-q > 1,
                             (sort-by     ; return sensitive value
                               last       ; with highest frequency
                               >          ; in data-q
                               (frequencies (nth (trans data-q) (first att)))))))))))
```

Now, we define a breach of privacy as follows:

$$Breach(S, G_i^*) = \begin{cases} 1, & if \quad s_{max}(G_i) = s_{max}(G_i'), \\ 0, & otherwise. \end{cases}$$

Therefore, the privacy level of the privatized data set is,

$$P_1 = 100 \times IPR(T^*) = 1 - \frac{1}{|Q|} \sum_{i=0}^{|Q|} Breach(S, G_i^*). \tag{4.4}$$

$IPR(T^*)$ stands for *increased privacy ratio* and has some similarity to A_{acc} of Brickell and Shamtikov [35], where $IPR(T^*)$ measures the attacker's ability to infer the sensitive attribute value of a target in the privatized data set T' compared to a baseline of the original data set T. To be

more precise, $IPR(T^*)$ measures the percent of total queries that did not cause a privacy *Breach*.

We baseline our work against the original data set (our worst case scenario) which offers no privacy and therefore its $IPR(T) = 0$. In our case, to have perfect privacy (our best case scenario) we create a privatized data set by simply removing the sensitive attribute values. This will leave us with $IPR(T') = 1$.

Figure 4.3 shows an example of how IPR works with three queries (Q1, Q2, and Q3). When the query is applied to the original and the privatized data, if the both report the same sensitive attribute value then we consider this a privacy breach. From the results of three queries we find that we are able to protect our data from 33% of the queries. Also, Listing 4 shows the source code for the main function of IPR.

Queries	Original	Privatized	Privacy Breach
Q1	0	0	yes
Q2	0	1	no
Q3	1	1	yes
			yes = 2/3 **IPR** = 1- 2/3 = 0.33

Figure 4.3: Example of how IPR is calculated based on queries.

4.3.2 Upper and Lower Bounds on IPR

When used in conjunction with instance selection algorithms like LeaF and CLIFF, Equation 4.4 is a *lower bound* on privacy. Recall that:

- From data set, D, CLIFF and LeaF discards X rows;
- Equation 4.4 is applied to the remaining $D - X$ rows.

Since data from the X discarded rows are never shared, it is fully private. Therefore, an upper bound on privacy is:

Listing 4 Source code for IPR main function.

```
(defn ipr
 "Returns IPR score for each sensitive attribute.
  Arguments:
  original data matrix
  private data matrix

  Options:
  :sav (default [10 20])
  vector containing the sensitive attribute value(s) and the class attribute is last.
  :n (default 10)
  used in equal frequency binning to create n number of bins.
  :query-size (default 1)
  used as amount of info attacker knows, can use 1, 2 and 4.
  :num (default 1000)
  can generate up to 1000 queries."
 [original-data
  private-data
  & {:keys [sav n query-size num worst]
     :or {sav [10 20] n 10 query-size 1 num 1000}}]
 (let [mybin  (bind-columns ; Bin data to make queries.
                (trans (butlast (efb2 original-data n)))
                (sel original-data :cols (dec (ncol original-data))))
       que    (take num (shuffle (get-queries mybin query-size sav :num num)))
       ans    (bind-rows              ; For each query
                (query-scores         ; find best guess sensitive value for
                  que                 ; original data.
                  mybin
                  original-data
                  original-data
                  query-score3
                  (butlast sav)
                  query-size)
                (query-scores         ; For each query
                  que                 ; find best guess sensitive value for
                  mybin               ; private data.
                  original-data
                  private-data
                  query-score3
                  (butlast sav)
                  query-size))
       scores (get-accs (sel ans :cols (range 3 (ncol ans))) (count que))]
   scores))   ; Return percent of best guesses that don't match.
```

$$P_2 = \frac{X}{D} + \frac{(D-X)}{D * P_1/100} \tag{4.5}$$

where P_1 comes from Equation 4.4. For example, if CLIFF and LeaF discard 80% of the data and, on the remaining data, we achieve an IPR of 80%. The resulting increased privacy is hence $0.8 + 0.2 * 0.8 = 96\%$.

Note that P_2 is an *upper bound* on privacy since it is possible that the patterns in the discarded data might repeat in the cached data. That said, given a large enough community sharing their data, there would always be some doubts about which members of the community had the exact values found in particular query. In this dissertation, we use the lower bound IPRs to show the worst-case results.

4.3.3 Query Generator

A *query generator* is used to provide a sample of attacks on the data. Before discussing the query generator, a few details must be established. First, to maintain some "truthfulness" to the data, a selected sensitive attribute(s) and the class attribute are not used as part of query generation. Here we are assuming that the only information an attacker could have is information about the non-sensitive attributes in the data set. As a result these attribute values (sensitive and class) are unchanged in the privatized data set.

To illustrate the application of the query generator, we use Table 4.2a and Table 4.2b. First, *equal frequency binning* is applied to the original data in Table 4.2a to create the sub-ranges shown in Table 4.2b. Next, to create a query, we proceed as follows:

1. Given a query size (measured as the number of {attribute sub-range} pairs), for this example we use a query size of 1;

2. Given the set of attributes A = (wmc, dit, noc, cbo, rfc, lcom, ca, ce);

3. Randomly choose an instance from the data, for this example we will use *row 1* in Table 4.2b, randomly select an attribute from A, e.g. *wmc* pair with its sub-range (6-14].

In the end the query we generate is, *wmc* = (6-14]. Table 4.3, shows more examples of queries, their sizes, the number and rows they match from the data set. We continue this process until we have used all instances or 1000 queries. Listing 5 shows the source code for generating queries from one instance.

Table 4.2: Example defect data for generating queries. First data is binned using equal frequency binning to create subranges. Table 4.2a shows the original data while Table 4.2b shows the data after equal frequency binning.

(a) Partial ant-1.3 Defect Data

#	wmc	dit	noc	cbo	rfc	lcom	ca	ce	loc	class
1	11	4	2	14	42	29	2	12	395	0
2	14	1	1	8	32	49	4	4	297	1
3	3	2	0	1	9	0	0	1	58	0
4	12	3	0	12	37	32	0	12	310	0
5	6	3	0	4	21	1	0	4	136	0
6	5	1	5	12	11	8	11	1	59	0
7	4	2	0	3	16	0	0	3	59	0
8	14	1	0	24	63	63	20	20	822	1

(b) ant-1.3 After Equal Frequency Binning

#	wmc	dit	noc	cbo	rfc	lcom	ca	ce	loc	class
1	(6-14]	[1-4]	[0-5]	(8-24]	(21-63]	(8-63]	(2-20]	(4-20]	(136-822]	0
2	(6-14]	[1-4]	[0-5]	[1-8]	(21-63]	(8-63]	(2-20]	[1-4]	(136-822]	1
3	[3-6]	[1-4]	[0-5]	[1-8]	[9-21]	[0-8]	0	[1-4]	[58-136]	0
4	(6-14]	[1-4]	[0-5]	(8-24]	(21-63]	(8-63]	0	(4-20]	(136-822]	0
5	[3-6]	[1-4]	[0-5]	[1-8]	[9-21]	[0-8]	0	[1-4]	[58-136]	0
6	[3-6]	[1-4]	[0-5]	(8-24]	[9-21]	[0-8]	(2-20]	[1-4]	[58-136]	0
7	[3-6]	[1-4]	[0-5]	[1-8]	[9-21]	[0-8]	0	[1-4]	[58-136]	0
8	(6-14]	[1-4]	[0-5]	(8-24]	(21-63]	(8-63]	(2-20]	(4-20]	(136-822]	1

For each query size, we generate up to 1000 queries because it is not practical to test every possible query. With these data sets the number of possible queries with arity 4 and no repeats is *38,760,000*.[2]

Each query must also satisfy the following *sanity checks*:

[2] $\frac{n!}{k!(n-k)!} = \binom{n}{k}$, where n is 19 (all attributes minus the class and sensitive attributes), and $k = 4$. This gives $3,876$ combinations. Four queries over a variable with ten values (i.e. the 10 values generated by EFB) generates a space of 10^4 options. Therefore the total number of possible queries of arity four is $10^4 * 3876 = 38,760,000$.

Table 4.3: Example: Queries, Query Sizes and the number of rows that match the queries, |G|. Table 4.2b is used for this example.

Query	Size	|G|	Row#
cbo = (8-24]	1	4	1,4,6,8
cbo = (8-24] and wmc = (6-14]	2	3	1,4,8
cbo = (8-24] and wmc = (6-14] and noc = [0-5] and ca = 0	4	1	4

- They must not include attribute value pairs from either the designated sensitive attribute or the class attribute;
- They must return at least one instance after a search of the original data set;
- They must not be the same as another query no matter the order of the individual {attribute sub-range} pairs in the query.

Listing 5 Source code for the query generator, with one instance.

```
(defn get-queries-one
 "Returns queries generated from one instance.
  Arguments:
  one instance from binned original data.
  all possible combinations of qids of size n.
  vector of sensitive attributes plus the class attribute (sav)."
 [one n sav]
 (let [atts   (range (count one))
       qids   (remove (set sav) atts) ; Get the qids.
       combos (combinations qids n)   ; Combos of qids of size n
       query  (fn [one combo]  ; For each combo, map qids to their values.
                (apply vector (map #(vector % (nth one %)) combo)))]
   (apply vector (map #(query one %) combos)))) ; Return qid, value pairs (queries)
```

4.3.4 IPR Evaluation

In previous work [11, 12] we calculated the IPR of a single sensitive attribute, specifically, *lines of code (LOC)*. In this section we evaluate IPR with different and multiple sensitive attributes. This

is done to show that IPR is a relevant privacy metric for data owners particularly since they are the ones that determine what the sensitive attributes are for their data.

To complete this evaluation of IPR we use the *Arc* defect data set and apply CLIFF&MORPH, selecting 20% of the data with CLIFF and randomly MOPRHing the selected data at a 30% difference for each attribute value to the nearest unlike neighbor. We do this because the aim here is only to show the versatility of IPR and it's independence from the different privacy models that could be applied to the data. We then select five attributes as sensitive and add LOC since it is used in previous work [11, 12]. This evaluation is repeated in Chapter 6 for three proprietary data sets.

IPR accepts as input, the original data and its privatized data candidate. We report on two possible cases that data owners can find the IPR of the sensitive attributes in their data if the privatized data candidate is to be published. The first case considers if the data owner determines that there is only one sensitive attribute in their data. Table 4.4 shows the IPRs of six different "sensitive" attributes as a function of an attacker's background knowledge represented as *query size*. Also, the *Bin?* column indicates whether or not the attribute values required equal frequency binning.

In the second case we consider the data owner with multiple (three) sensitive attributes in their data. To calculate the IPR for multiple sensitive attributes we determine the IPR of each attribute and report the average. Table 4.5 shows the IPRs for six randomly chosen triplets of sensitive attributes.

The general trend in these results is that, as the attacker's background knowledge increases, the sensitive attribute(s) of the data are better protected. This is counter-intuitive, however we conjecture that since the data is changed with MORPH and it is possible that a target not be present in the published data, as the attacker's knowledge increases it is less likely that they find their target.

From these results we also find that IPR is dependent on the "sensitive attribute" chosen. For instance, from Table 4.4, wmc, rfc and loc in the gray rows, have relatively higher IPRs than dit, noc and mfa.

Table 4.4: Case 1: IPRs of different sensitive attributes in the Arc defect data set. The gray rows indicates those with relatively higher IPRs (when the query size is one) than the other rows at query size=1.

Sensitive Attributes	Bin?	Query Size	Quartiles		
			25%	50%	75%
wmc	TRUE	1	75.0	75.0	75.0
		2	87.6	88.1	88.5
		4	90.0	91.1	91.6
dit	FALSE	1	50.8	50.8	50.8
		2	70.5	70.9	71.3
		4	73.3	74.1	75.1
noc	FALSE	1	43.9	43.9	43.9
		2	62.4	63.1	63.9
		4	67.5	68.6	69.3
rfc	TRUE	1	70.4	70.4	70.4
		2	86.2	86.6	87.3
		4	90.0	90.4	90.8
loc	TRUE	1	74.8	74.8	74.8
		2	87.9	88.5	89.0
		4	91.8	92.3	93.0
mfa	TRUE	1	47.5	47.5	47.5
		2	66.6	67.4	69.4
		4	68.9	70.4	71.1

In conclusion, IPR provides a data owner with information required to decide if to publish data with or without a particular sensitive attribute(s). Also, it is important to note that in our work attributes that are not considered as sensitive, their values are transformed via MORPH and what IPR tells us is how protected the sensitive values will be if the data is published with the sensitive attributes untouched while the other attribute values are changed as a means to "hide" the sensitive attribute values. We expand on this notion later on in Section 6.3.1, where we calculate IPRs for sensitive attributes that are MORPHed.

Table 4.5: Case 2: IPRs of different groups of sensitive attributes in the Arc defect data set.

Sensitive Attributes	Query Size	Quartiles		
		25%	50%	75%
wmc:dit:rfc	1	64.1	64.1	64.1
	2	79.7	80.2	80.6
	4	84.3	84.4	85.3
wmc:noc:rfc	1	63.1	63.1	63.1
	2	80.0	80.5	80.8
	4	84.5	85.0	85.9
wmc:loc:mfa	1	65.0	65.0	65.0
	2	79.8	80.5	80.8
	4	82.9	83.5	84.0
wmc:rfc:mfa	1	65.7	65.7	65.7
	2	79.9	80.8	81.3
	4	84.3	84.4	84.9
dit:noc:mfa	1	46.2	46.2	46.2
	2	68.6	69.3	70.2
	4	72.2	72.7	73.6
dit:noc:loc	1	56.1	56.1	56.1
	2	75.3	76.4	77.3
	4	81.3	81.8	82.2

4.4 Summary

LACE is made to defend against the privacy threat of sensitive attribute disclosure. It creates privatized data candidates through minimization and then obfuscation of data in order to facilitate data sharing. CLIFF is the algorithm used for minimization and MORPH for obfuscation. To facilitate collaboration among multiple data owners we use the leader follower algorithm, LeaF. We conjecture, that LACE can provide data owners with a viable means to protect the confidentiality of their data. To measure the protection offered through LACE, we find the IPR which models a sensitive attribute disclosure attack using background knowledge. Finally, the usefulness of privatized data can be measured in various ways, such as information loss, distance of distributions, aggregate query answering or classification. In this dissertation we measure utility in terms of classification via cross project defect prediction.

In subsequent chapters we evaluate LACE based on a single data owner usage (Chapter 5) then based on multiple data owners (Chapter 6). We start by evaluating CLIFF&MORPH for single usage and baseline this with other standard privacy algorithms.

Chapter 5

Experiment 1: Comparison of CLIFF&MORPH with other Privacy Algorithms

5.1 Introduction

The original intent of this chapter was to show that the parameters we offered in our paper [12] for CLIFF&MORPH were satisfactory over a large number of data sets. Our preliminary experiments showed that this was not the case, a result that was somewhat discomforting since that meant we were now forced to conduct *parameter tuning* experiments, to offer a more comprehensive comparison of CLIFF&MORPH with other privacy algorithms.

Parameter tuning is the black art of adjusting the "magic numbers" within an algorithm in order to improve that algorithm. This section documents the multiple problems other researchers have encountered with tuning. The rest of this chapter is more optimistic- we show that for the particular case of privacy algorithms for software engineering, that (a) the effort associated with parameter tuning is not excessive; and (b) the tunings can transfer to other domains (which means that tuning

does not need to be repeated for each new domain).

Work in parameter tuning tends to focus on setting parameters for evolutionary algorithms (EAs) [124–126], search-based software engineering (SBSE) techniques [127–129] or tuning data mining algorithms. Considering tuning data mining algorithms and software engineering, a common and critical problem is that most algorithms use a variety of parameters, which are domain and data specific. Reusing a set of parameter values from one application to another usually leads to poor results. For example, Bayes classifiers use the "M" and Laplace factors to handle low frequency counts. Standard default values are M=2 and L=1, [130] but there is no telling what is the best configuration for a particular domain. These standard parameter values are usually derived empirically from applications in domains outside software engineering. For example, default parameter values used by many information retrieval techniques are established based on data from text retrieval tasks on natural language corpora. Recent empirical research in applications of text retrieval in software engineering [131–133] showed that best results are in fact obtained for configurations different from those used in natural language retrieval applications. More than that, the quality of the results is highly dependent on these configurations [134, 135]. The observation also holds for many defect prediction approaches [69]. For most data mining algorithms, the space of possible parameter values is extremely large. Finding the right combination is often based on intuition rather than on science.

Eiben et al. [126] explain two categories of tuning methods:

1. non-iterative and

2. iterative tuners.

According to the authors, all tuning methods work by the GENERATE-and-TEST principle where each parameter vector is generated and their performance tested by executing the EA or in our case, a privacy algorithm. The non-iterative methods execute the GENERATE step once, creating a set of parameter vectors that are tested during the TEST step to find the "best param-

eter vector" in the set. Here the GENERATE step can be done via random sampling (our choice in this work), generating a systematic grid in the parameter space, or some space filling set of vectors. Iterative tuning methods start with a small initial set and create new vectors iteratively during execution [126]. These methods are commonly done via meta-EAs and Iterative Sampling Methods.

Like many algorithms, privacy algorithms come with at least one parameter that needs to be set. Considering that different parameter settings can cause large variation in performance [136], this places a lot of the burden on the data owner, whose only concern is whether or not to release data based on the algorithm performance of both high privacy and utility.

Another concern lies with the idea of a generalist result. Many of these privacy algorithms may come with recommended settings which the author(s) certified as able to work well on multiple and different problems. However, since the creator of the algorithm does not know about all the problems their algorithm will face, it may still be the case that the recommended parameter vectors will perform worse on other unseen problems. This idea has resulted in the "no-free-lunch" theorem for optimization [137] which states that for any algorithm, any elevated performance over one class of problems is offset by performance over another class. In other words algorithms performing well on one type of problem, will perform worse on another [125].

Recall that parameter tuning works by means of a generate-test method. In the case of privacy algorithms which have two objectives, during the TEST step, the performance is measured by the level of privacy offered and the utility from (say) defect prediction. In the end, the "best parameter vector" is the one with the best (or good enough) performance of high privacy and high utility.

The historical evidence is that parameter tuning can be very hard, possibly even needing state-of-the-art multiple objective optimizers [124]. For example, in standard tuning methods, associated with evolutionary algorithms, it is common practice to set a simulation limit to 1000. As a result, parameter tuning requires a lot of computer power [124]. In addition, from their results Smit et al. [124] found that overall, it was difficult for tuners to consistently reach areas with the best EA

setup that offered a good solution. From this they concluded that the tuning algorithms they studied needed to be ran several times in order to find a good parameter vector. The point of this chapter now is a "good news" result that, at least for privacy and defect prediction, we did not need to go to these extremes to find our "magic numbers".

5.2 Experimental Setup

As explained, in this chapter we investigate the performance of CLIFF&MOPRH compared with other privacy methods specifically *k*-anonymity and data swapping, using parameter tuning experiments. The following sections explain our experimental setup, including the data sets and the defect predictor used.

5.2.1 Data

A total of 10 open-source static code defect data sets are used in this study. These data sets are the latest open-source releases from Table 3.1. They were collected and shared by Jureczko et al. [15,81]. As explained in Section 3.7.1, each instance in a data set represents a class and consists of two parts: 20 independent static code attributes and the dependent attribute labeled "defects" indicating the number of defects in the class. For our work, we refer to each class as an instance. Additionally, instances with no defects are labeled as *0*, and instances with one or more defects are labeled as *1*. Below we present short descriptions of the open-source Java projects and Table 5.1 shows the names and details of the data sets used in these experiments, the type (open-source or proprietary), the number of instances in each data set, and the number and percent of defects. In addition, recall that Table 2.1 describes the attributes of these data sets.

Short descriptions of the open-source Java projects are presented as follows:

- **Ant** [138], is a well known, java-based, shell independent, build tool. Ant is mainly used

to build java applications. With Ant, developers can compile, assemble, test and run Java applications.

- **Camel**: According to [139], the Camel framework is a routing-engine builder that allows you to define your own routing rules, decide from which sources to accept messages, and determine how to process and send those messages to other destinations. Camel uses an integration language that allows you to define complex routing rules, akin to business processes. It also allows the developer an opportunity to integrate any kind of system, without the need to convert data to a canonical format.

- **Ivy** [140]: is a tool for managing (recording, tracking, resolving and reporting) project dependencies.

- **jEdit** [141]: is a cross platform programmer's text editor written in Java.

- **Lucene** [141, 142]: provides Java-based indexing and search technology as well as spell-checking, hit highlighting and advanced analysis/tokenization capabilities.

- **Poi** [143]: creates and maintains Java APIs for manipulating various file formats based upon the Office Open XML standards (OOXML) and Microsoft's OLE 2 Compound Document format (OLE2). In short, according to the project website [143], one can read and write MS Excel files using Java. In addition, you can read and write MS Word and MS PowerPoint files using Java.

- **Synapse** [144]: According to the project website [144], Synapse is a lightweight and high-performance Enterprise Service Bus (ESB). Powered by a fast and asynchronous mediation engine. It provides exceptional support for XML, Web Services and REST. In addition to XML and SOAP, Apache Synapse supports several other content interchange formats, such as plain text, binary, Hessian and JSON. The wide range of transport adapters available for Synapse, enables it to communicate over many application and transport layer protocols.

Apache Synapse supports HTTP/S, Mail (POP3, IMAP, SMTP), JMS, TCP, UDP, VFS, SMS, XMPP and FIX.

- **Velocity** [141, 145]: a Java-based template engine that permits anyone to use a powerful template language to reference objects defined in Java code. From their description, Jureczko et al. [141], go on to explain that Velocity separates Java code from web pages, making the websites more maintainable over a lifespan and providing a viable alternative to Java Server Pages or PHP, a server-side scripting language for web development.

- **Xalan** [146]: is an XSLT (Extensible Stylesheet Language Transformations) processor for transforming XML documents into HTML, text, or other XML document types.

- **Xerces** [147]: is a parser that supports XML 1.0 recommendation, and contains advanced parser functionality such as support for the W3C's XML Schema recommendation version 1.0, DOM Level 2 version 1.0, and SAX Version 2, in addition to supporting the industry-standard DOM Level 1 and SAX version 1 APIs. Some applications for Xerces include: building XML-savvy Web servers, the next generation of vertical applications which will use XML as their data format, on-the-fly validation for creating XML editors, ensuring the integrity of e-business data expressed in XML, and building truly internationalized XML applications.

5.2.2 Benchmark Privacy Algorithms

In order to benchmark our approach, we compare it against data swapping and k-anonymity. Recall that data swapping is a standard perturbation technique used for privacy [5, 42, 63]. This is a permutation approach that de-associates the relationship between a non-sensitive quasi-identifier and a numerical sensitive attribute. In our implementation of data swapping, for each quasi-identifier a certain percent of the values are swapped with any other value in that quasi-identifier. For our experiments, these percentages are 10%, 20%, 40% and 80%.

Our implementation of k-anonymity follows the Datafly algorithm [34, 60] for k-anonymity. Recall from Section 2.4, that the algorithm starts with the input of a set of quasi-identifiers, k, and a generalization hierarchy. Datafly replaces values in the quasi-identifiers according to the hierarchy. This generalization continues until there are k or fewer distinct instances which are suppressed.

Table 5.1: Objective Data Sets for Open-source Project Data

Defect Data	**Type**	**# Instances**	**# Defects**	**% Defects**
ant-1.7	open-source	1066	166	15.6
camel-1.6	open-source	1252	188	15.0
ivy-2.0	open-source	477	40	8.4
jEdit-4.1	open-source	644	79	12.3
lucene-2.4	open-source	536	203	37.9
poi-3.0	open-source	531	281	52.9
synapse-1.2	open-source	269	86	32.0
velocity-1.6	open-source	261	78	29.9
xalan-2.6	open-source	1170	411	35.1
xerces-1.3	open-source	545	69	12.7

5.2.3 Naive Bayes

We use Naive Bayes as our defect predictor based on an extensive study done by Lessmann et al. [66] who found that Naive Bayes performs well compared to more complex predictors. In addition, another study by Menzies et al. [69] endorsed Naive Bayes for building defect predictors since it performed better than the other learners used in their study, namely J48 and OneR.

Lewis [83] describes Naive Bayes as a classifier based on Baye's rule shown in Equation 5.1. It is a statistical based learning scheme which assumes that attributes are equally important and

statistically independent.

$$P(c_k|x) = P(c_k) \times \frac{P(x|c_k)}{P(x)} \tag{5.1}$$

where c_k is a member of the set of values for the dependent attribute. In our case c_k could be *0* or *1*. Also, x represents a test instance or unknown instance. So, to classify a test instance, Naive Bayes finds the conditional probability of that instance being labeled c_k. The c_k with the highest probability is chosen as the label for x. We do not implement Naive Bayes ourselves, instead we use the Weka [148] implementation with the default values set.

5.2.4 Performance Evaluation

Privacy algorithms have two performance objectives: 1) privacy and 2) utility. To measure the privacy offered by a privacy algorithm for a specific data set we use Increase Privacy Ratio or IPR. A full explanation of IPR and how it works is in Section 4.3. IPR measures the sensitive attribute disclosure risk for specific attributes of a privatized data candidate by comparing query results with the original data. In this chapter we focus on *lines of code* as the sensitive attribute. Other sensitive attributes will be explored in the next chapter.

The size of a query represents the amount of background knowledge an attacker has about a target record in a data set. In the experiments of this dissertation we use a query size of 1 due to the analysis in Section 4.3.4, where when *query size = 1*, sensitive attribute value disclosure is worse than for higher query sizes.

To measure the utility of a privatized data set, we measure the performance of defect predictors. The performance measures used for the defect predictors are shown in Table 5.2 and summarized as follows:

- Recall or *pd* is equal to how much of the target (defective instances) are found. The higher the pd, the fewer the false negative results.

- Probability of false alarm or *pf* measures how many of the instances that triggered the detector actually did not contained the target (defects) concept. Like pd, the highest pf is 100% however its optimal result is 0%.

- *g*-measure (harmonic mean of pd and 1-pf): Instead of the *f*-measure, we report on the *g*-measure. The 1-pf represents *Specificity* (not predicting instances without defects as defective. *Specificity* (1-pf) is used together with *pd* to form the *G-mean*$_2$ measure seen in Jiang et al. [86]. It is the geometric mean of the pd's for both the majority and the minority class. In our case, we use these to form the *g*-measure which is the harmonic mean of pd and 1-pf.

Measures such as accuracy, precision, and f-measure are not shown in our experimental results since they are poor indicators of performance for data where the target class is rare (in our case, the defective instances). This is based on a study done by Menzies et al. [149] which shows that when data sets contain a low percentage of defects, precision can be unstable.

Table 5.2: Some popular measures used in software defect prediction work.

		Actual	
		yes	**no**
Predicted	**yes**	TP	FP
	no	FN	TN
Recall (pd)	$\frac{TP}{TP+FN}$		
pf	$\frac{FP}{FP+TN}$		
***g*-measure**	$\frac{\textbf{2*pd*(100-pf)}}{\textbf{pd+(100-pf)}}$		

5.3 Analysis 1. Does CLIFF&MORPH provide better balance between privacy and utility than other state-of-the-art privacy algorithms?

5.3.1 Design

Many privacy algorithms require that the data owner have some knowledge about how the algorithms work in order to use the appropriate parameters. Without the correct understanding it is possible to produce a privatized data candidate that is prone to malicious attacks. Therefore, a more substantial parameter tuning experiment is required to compare different privacy algorithms. We use the following experimental design to answer our first research question: given different parameter values, "Does CLIFF&MORPH provide better balance between privacy and utility than other state-of-the-art privacy algorithms?"

From Table 5.1, we use ant-1.7 as the training set and jEdit-4.1 as the test set. We first privatize ant-1.7 in 24 ways using the three privacy methods, with different parameter vectors. We then calculate their IPRs and the utility of the privatized data through CPDP with jEdit-4.1. These data sets are chosen and assigned arbitrarily to showcase the possible variance in the performance. In a later experiment looking at the generalizabilty of our parameter values, we consider more data from Table 5.1.

Table 5.3, shows the parameters and the values used in this experiment and Algorithm 3 shows the pseudo-code for the experimental design. Note that CLIFF&MORPH has two parameters to set (r and p). Here r is a random value that determines the border between an original instance and it's nearest unlike neighbor (the instance with a different defect label). In previous work [11, 12], r was selected from a range of 0.15 to 0.35. Our intuition was that to establish a privatized data set that was both private and useful, we would change the data so that it was closer to the original instance than to it's nearest unlike neighbor. In this experiment we increase that range to 0 to 1.

Table 5.3: Privacy algorithms and their parameter values used in this study.

	Parameters and Values			
Privacy Algorithms	**r**	**p**	**k**	**q**
CLIFF&MORPH	0...1	{0.1, 0.2, 0.4}	NA	NA
Swapping	NA	{0.1, 0.2, 0.4, 0.8}	NA	NA
K-Anonymity	NA	NA	{2, 4, 8, 16}	2...15

Algorithm 3 Pseudo-code of the experimental design for Analysis 1.

Input
Data: Train ant-1.7, Test jEdit-4.1
Algorithms: 1 CLIFF&MORPH, 2 Data Swapping, 3 k-anonymity
Parameters: r (0 to 1), p (0.1 0.2 0.4 0.8), k (2, 4, 8, 16), q (2 to 15)
Iterations: i = 24

Output
24 rows of {algorithm, ipr, g, r, p, k, q}

while $i \geq 1$ **do**
  if Rand$(Algorithms) = 1$ **then**
    CLIFF&MORPH$(Rand(r), Rand(p = [0.2 to 0.8])) \mapsto \{algorithm, ipr, g, r, p, k, q\}$
  else if Rand$(Algorithms) = 2$ **then**
    Data Swapping$(Rand(p - 1.0)) \mapsto \{algorithm, ipr, g, r, p, k, q\}$
  else
    ***k*-anonymity**$(Rand(k), Rand(q)) \mapsto \{algorithm, ipr, g, r, p, k, q\}$
  end if
  Decrement(i)
end while

For CLIFF&MORPH, p represents the proportion of data kept after applying CLIFF. Data Swapping has one parameter to set, p. Here p represents the proportion of attribute values randomly chosen to be swapped with any other of that attribute's values. Last, k-anonymity has two parameter values to set, k and q. Values for k indicate that for the quasi-identifiers in a group, each instance is the same as k-1 other instances in the data set. Values for q represent the number of quasi-identifiers in the data set. Note that when a parameter is not applicable to a privacy method, *NA* is returned.

5.3.2 Results and Discussion

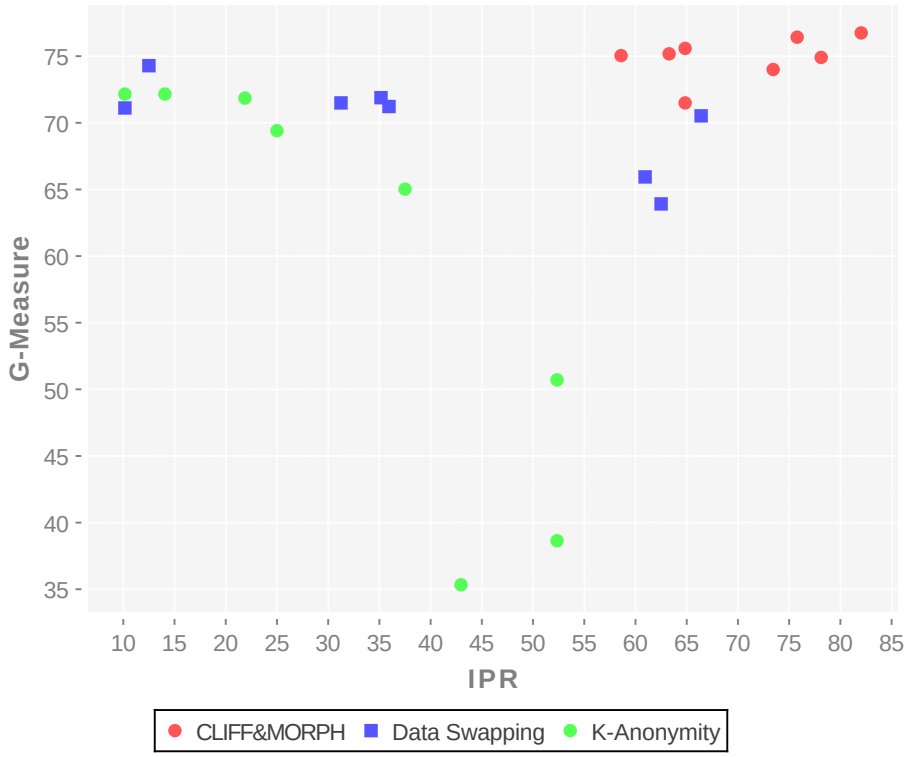


Figure 5.1: The IPRs and *g*-measures of CLIFF&MORPH, *k*-anonymity and data swapping. In this figure, an ideal method would have results at the top-right. CLIFF&MORPH outperforms both *k*-anonymity and data swapping with higher IPRs and *g*-measures. All algorithms show a wide variance in the IPR results while only *k*-anonymity also shows variance in the *g*-measures, decreasing as privacy (IPR) increases.

Figure 5.1 and Table 5.4 show the results of the IPRs and *g*-measures for 24 parameter tuning experiments for the privacy algorithms used in this study. From Figure 5.1 it is clear that CLIFF&MORPH outperforms both *k*-anonymity and data swapping with higher IPRs and *g*-measures. All the algorithms show a variance in the IPR results while only *k*-anonymity *also* shows variance in the *g*-measures, decreasing as privacy (IPR) increases. From Table 5.4 we also see the parameter values used for each experimental run. These results indicate that the parameter setting for the CLIFF&MORPH algorithm are less prone to large variance in the IPR and *g*-measures. However as the parameter settings changes for data swapping and *k*-anonymity, both show variance in IPR

Table 5.4: The results for 24 experimental runs with their parameter values used to find the *g*-measures and the IPRs.

Algorithms	**r**	**p**	**k**	**q**	**ipr**	**g**
CLIFF&MORPH	0.7	0.1	NA	NA	78.1	74.9
	1.0	0.1	NA	NA	82.0	76.7
	0.4	0.4	NA	NA	64.8	75.6
	0.7	0.4	NA	NA	64.8	71.5
	0.9	0.2	NA	NA	75.8	76.4
	0.5	0.1	NA	NA	73.4	74.0
	0.2	0.4	NA	NA	63.3	75.2
	0.1	0.4	NA	NA	58.6	75.0
Data Swapping	NA	0.4	NA	NA	31.3	71.5
	NA	0.1	NA	NA	12.5	74.3
	NA	0.8	NA	NA	66.4	70.5
	NA	0.8	NA	NA	60.9	65.9
	NA	0.8	NA	NA	62.5	63.9
	NA	0.4	NA	NA	35.2	71.9
	NA	0.1	NA	NA	10.2	71.1
	NA	0.4	NA	NA	35.9	71.2
K-Anonymity	NA	NA	8	9	43.0	35.3
	NA	NA	2	15	52.3	50.7
	NA	NA	2	6	21.9	71.9
	NA	NA	2	4	14.1	72.2
	NA	NA	2	3	10.2	72.2
	NA	NA	4	5	25.0	69.4
	NA	NA	16	9	52.3	38.6
	NA	NA	4	8	37.5	65.0

with data swapping ranging from 10.2% to 66.4% and *k*-anonymity ranging from 10.2% to 52.3%. In addition *k*-anonymity also shows a large variance in *g*-measures ranging from 35.3% to 72.2%.

This result for *k*-anonymity is consistent with work done by Grechanik et al. [3] where increasing privacy by increasing *k* decreased their utility which they measured as test coverage. In addition, the high *g*-measures for data swapping can be as a result of instances maintaining certain distribution properties that allow it to be more effective in preserving utility than privacy algo-

rithms that use data suppression and data generalization like k-anonymity [5]. Finally, we attribute the high IPRs for CLIFF&MORPH to instance selection with CLIFF which offers 100% privacy to those instances not selected for sharing and the high g-measures and IPRs to MORPH which insures that the remaining instances are mutated to the point that their class labels do not change.

From these results, we can now answer RQ1, Does CLIFF&MORPH provide better balance between privacy and utility than other state-of-the-art privacy algorithms? Our answer is:

> **A1**: For the various parameter values investigated, it is clear that CLIFF&MORPH offer the better balance for privacy and utility than both data swapping and k-anonymity. In addition, these results also indicate that in cases where privacy algorithms contain at least one parameter to set, parameter tuning is necessary for adequate comparison of privacy algorithms used in this study (some more than others).

5.4 Analysis 2. How hard is parameter tuning for privacy algorithms?

5.4.1 Design

In this analysis, we expand on the design of our first analysis from Algorithm 3 by increasing i to 192. In other words, we do a total of 192 runs of parameter tuning experiments for the three privacy algorithms studied in this work. We do this to see if the results stabilize and how long it takes to do so in terms of the number of experimental runs of the privacy algorithms.

5.4.2 Results

Figure 5.2 and Table 5.5 show the results of this experiment. From Figure 5.2, we see the stability of the performance of the privacy algorithms, CLIFF&MORPH, data swapping and k-anonymity.

Chart *a* shows the results of 24 runs as seen in our first analysis, *b* shows 48, *c* shows 96 and *d* shows 192 experiment runs.

The following general pattern holds as the number of runs increase. As before, the *k*-anonymity algorithm has large variance for both the IPR and *g*-measures. Data swapping shows large variance for IPR and minimal variance on *g*-measures. The CLIFF&MORPH algorithm with variance on IPR and higher g-measures than both *k*-anonymity and data swapping. This shows that finding a privatized data candidate that meets a data owner's standards for privacy and utility does not require an exhaustive search.

Table 5.5 offers a closer look at the results. For all 192 runs we sort them in descending order according to the harmonic mean of the IPR and the *g*-measure calculated as follows:

$$\frac{2 \times ipr \times g}{ipr + g} \tag{5.2}$$

Also included in Table 5.5 are the a, b, c and d columns from the charts in Figure 5.2. They represent the results of 24, 48, 96 and 192 experimental runs respectively. The checkmarks in these columns indicate the amount of runs required before each of the top 10 results are found. Also included in the last two rows of the table are the results for data swapping and *k*-anonymity, with the first column indicating where they rank in terms of the harmonic mean for 192 runs, 55 and 145 respectively.

Finally, the checkmarks in columns in *a*, *b*, *c* and *d* for each level of background knowledge show that after only 24 simulations, we were able to find at least one result in the top 10. Additionally, this number more than doubles after 48 simulations.

5.4.3 Discussion

From the results of our first analysis we found that parameter tuning is necessary when using privacy algorithms because of the large variance in IPR and *g*-measures. However we wondered

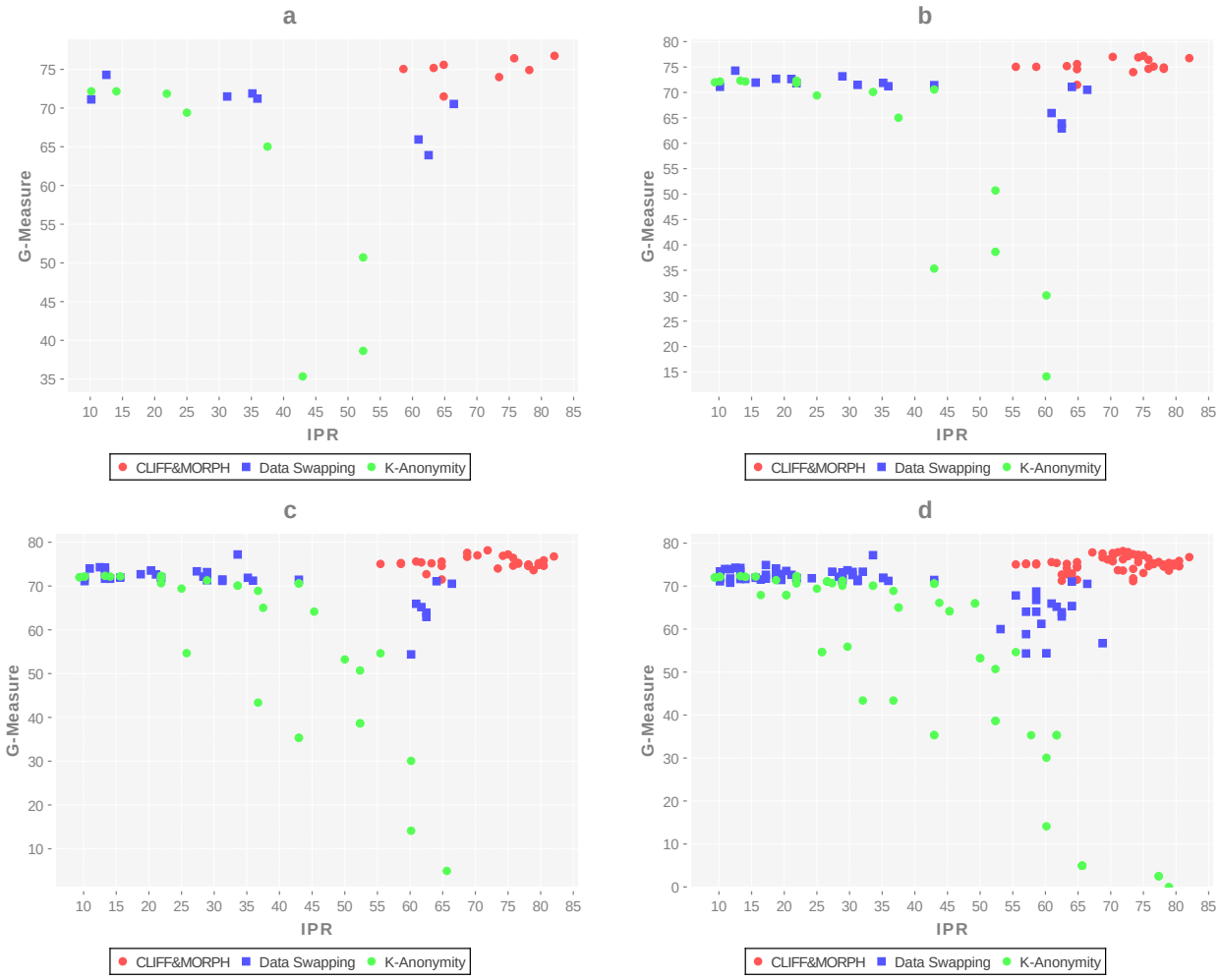


Figure 5.2: The stability of the performance of the privacy algorithms, CLIFF&MORPH, Data Swapping and *k*-anonymity. a) Shows the results of 24 simulations, b) shows 48, c) 96 and d) 192 runs. As seen, the general pattern holds as the number of runs increase. This shows that finding a privatized data candidate that satisfies a data owner's criteria is not exhaustive.

about the number of runs that would be required to find an adequate solution. To answer this question, for our second analysis we increased the number of runs to determine if the top results at 192 runs were present at 24, 48 and 96 runs.

The first thing to notice is that neither data swapping nor *k*-anonymity appear in the top 10 privatized data candidates. We conclude that for data swapping this is because the IPRs are low when the probability of swapping values are low. While for *k*-anonymity we conclude that this is

Table 5.5: This table shows the top 10 privatized data candidates sorted in descending order according to the harmonic mean between IPR and G-Measure. Only CLIFF&MORPH appears in the top 10 and all of them appeared after 48 runs of the algorithm. Data Swapping and K-Anonymity have the 55th and 145th harmonic means of 192 experimental runs.

	Algorithms	**IPR**	**G-Measure**	***Harmonic Mean***	**a**	**b**	**c**	**d**
1	CLIFF&MORPH	82.0	76.7	79.3	√			
2	CLIFF&MORPH	80.5	75.9	78.1			√	
3	CLIFF&MORPH	80.5	74.7	77.5				√
4	CLIFF&MORPH	80.5	74.6	77.4			√	
5	CLIFF&MORPH	79.7	75.2	77.4			√	
6	CLIFF&MORPH	79.7	74.8	77.2			√	
7	CLIFF&MORPH	78.9	75.4	77.1				√
8	CLIFF&MORPH	78.1	75.0	76.5				√
9	CLIFF&MORPH	78.1	74.9	76.5	√			
10	CLIFF&MORPH	77.3	75.6	76.5				√
55	Data Swapping	66.4	70.5	68.4	√			
145	K-Anonymity	49.2	66.0	56.4				√

because *k*-anonymity offers no protection against background knowledge [35] used for sensitive attribute disclosure attacks and so their IPRs generally tend to be low. In addition, on occasions when the IPRs are high due to high *k* and *q* values, their *g*-measures are low.

We also found that the top results for each privacy algorithms studied, can be found at 24 simulations. Based on the findings of our second analysis, we answer RQ2 (“ How hard is parameter tuning for privacy algorithms?”) as follows:

> **A2**: When seeking the best parameter values for privacy algorithms, the search does not have to be exhaustive. A *best* privatized data candidate can be found in a few runs, in this case 24.

5.5 Analysis 3. Are the results for parameter tuning for privacy algorithms useful for reducing the search budget?

Recall that conclusions made from previous analyses are based on the CPDP results using only two project defect data sets, *ant-1.7* as the training set and *jEdit-4.1* as the test set. In this section we explore the notion that the "best parameter values" learned from our second analysis can be used for other projects as well. Since the data owners' search ends when they are satisfied with a particular result, the ability to transfer parameter knowledge would reduce any search budget (i.e. number of runs).

With multiple privacy algorithms and even more possible parameter values, there is a concern about the search budget required to find the "best parameter values" for each algorithm. However, the goal of producing a privatized data candidate is not necessarily to find the absolute *best* result. That would require testing all the possible parameter vectors and their algorithms. Instead it is the data owners' who decide on what is their *best* result. This may by based on company guidelines or personal choice.

5.5.1 Design

In this experiment we want to find out if lessons learned from one project can be transferred to other projects to reduce the search budget. The lessons learned here are the "best parameter values" for the privacy algorithms studied here. Figure 5.3, Figure 5.4, and Figure 5.5 show the parameter values used in the previous analyses in this chapter, and the performance measured as the harmonic mean of IPR and the *g*-measure (see Equation 5.2). These figures help support our choices of *best* parameter values for each privacy algorithm.

For CLIFF&MORPH, based on Figure 5.3 we see that *r* which Class Boundary (r) used by the MORPH algorithm to determine how much the new synthetic instance should move to its nearest unlike neighbor. works well for all values in the range of 0 to 1, however we limit the range of

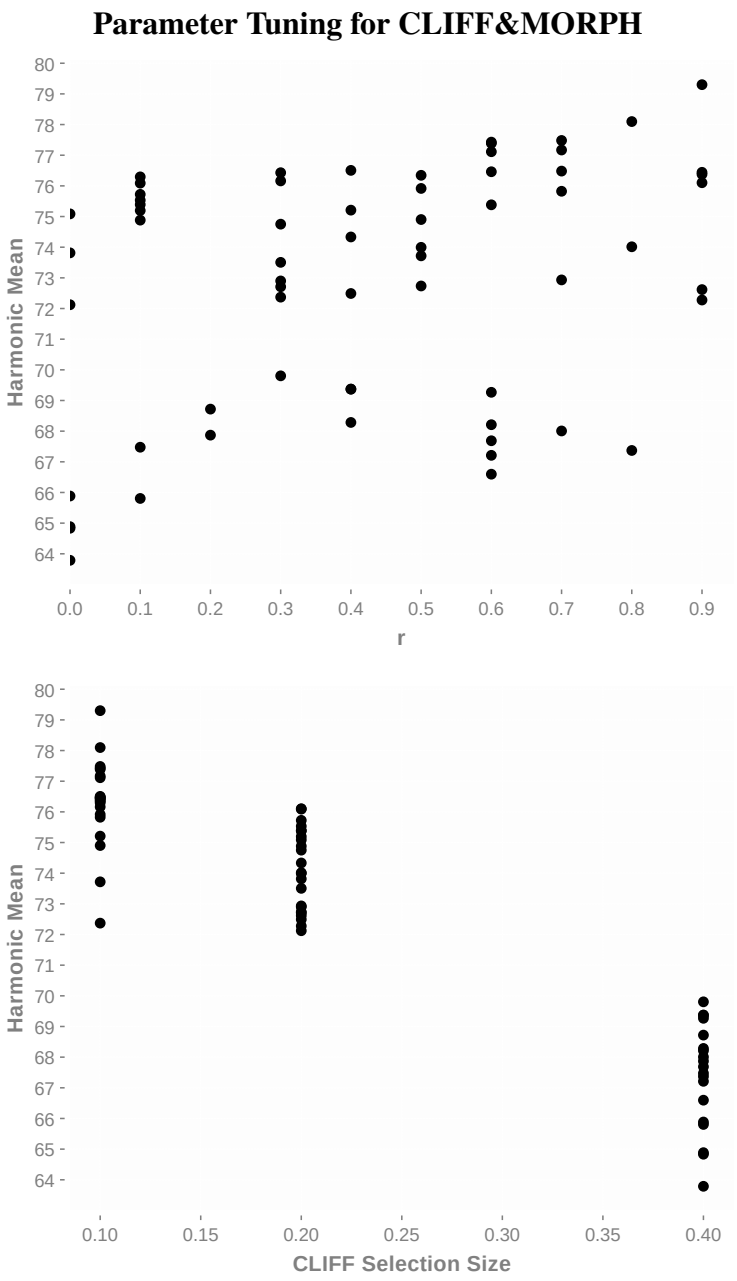


Figure 5.3: The parameters that allow CLIFF&MORPH to have the best performance. For the Class Boundary (r) used by the MORPH algorithm to determine how much the new synthetic instance should move to its nearest unlike neighbor. From this chart we choose a range of 0.3 to 1 for r. For CLIFF&MORPH, we choose p=0.1 and 0.2, i.e. after applying CLIFF, we choose 10% or 20% of the top ranked instances.

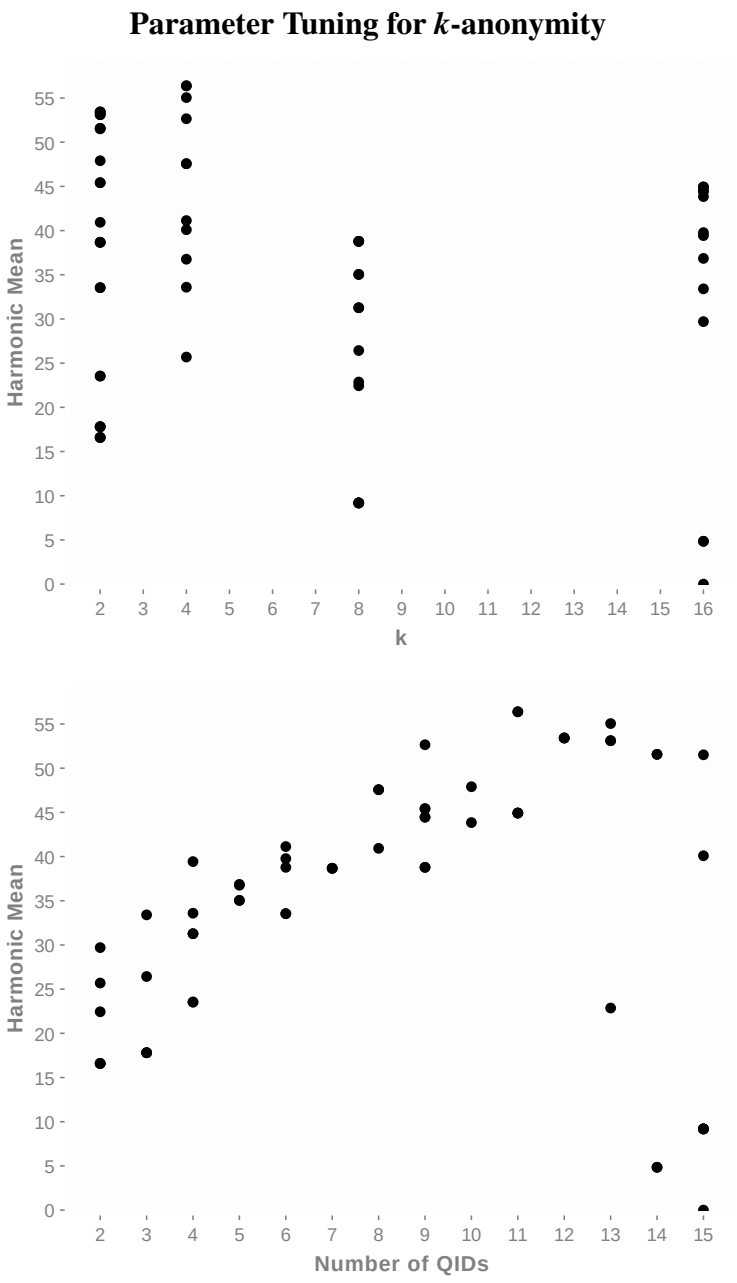


Figure 5.4: The parameters that allow *k*-anonymity to have the best performance. From chart on the top we choose k=2 an 4 and q is a range form 8 to 15.

r to 0.3 to 1 which contains the majority of the better results. Also for *p* which is the CLIFF selection size we decided on 0.1 and 0.2. For data swapping, the probability of swap (*p*) is 0.8, since Figure 5.5 shows it to have a much higher harmonic mean than the other three values. All

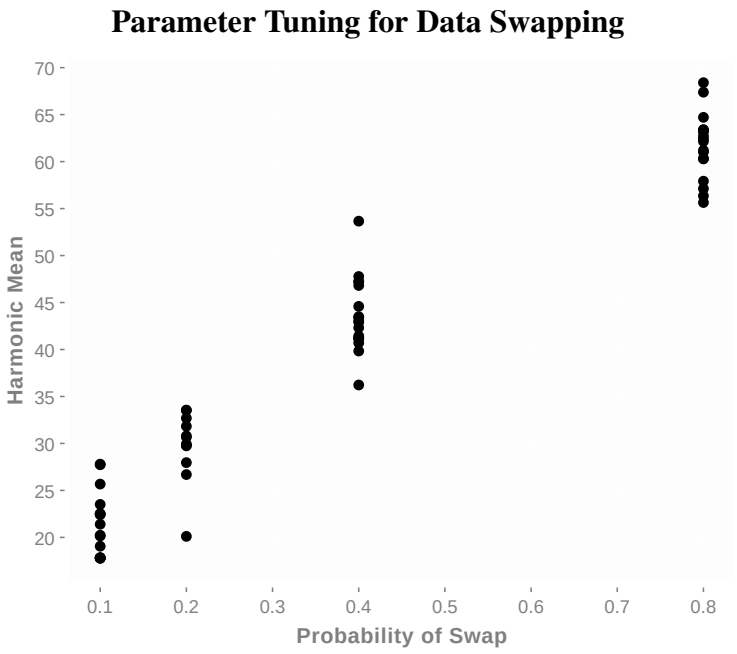


Figure 5.5: The parameters that allow data swapping to have the best performance. From this chart we choose 0.8 for the probability of swap(p).

harmonic means with 0.8 are at 55% and above while all others are 55% and below. Finally for *k*-anonymity, from Figure 5.4, *k* is set to two and four, while *q* the number of QIDs will have a range of 8 to 15.

We privatize the data sets and perform CPDP experiments. We benchmark this work with the CPDP of the original (non-privatized) data. The pseudo-code for the experimental design is shown in Algorithm 4 for CLIFF&MORPH. The same holds for the other algorithms and their parameter values as well.

5.5.2 Results

Table 5.6 shows the *g*-measures based on the "best parameter values" for CLIFF&MORPH, data swapping and *k*-anonymity from Analysis 2. For the test set we continue to use jEdit-4.1 as a constant in our analysis. What we find here is that there is no significant difference between each

Algorithm 4 Pseudo-code of the experimental design for Analysis 3.

Input
Data: ant-1.7, camel-1.6, ivy-2.0, jEdit-4.1, lucene-2.4, poi-3.0, synapse-1.2, velocity1.6, xalan-2.6, xerces-1.3,
Algorithm: CLIFF&MORPH
Parameters: r 0.3 to 1, p 0.1 or 0.2
Learner: Naive Bayes

Output and initial values
IPR $\leftarrow \emptyset$ {R}eturns the IPRs of privatized Data.
G $\leftarrow \emptyset$ {R}eturns the matrix of g-measures of Data from CPDP.
G$'$ $\leftarrow \emptyset$ {R}eturns the matrix of g-measures of privatized Data from CPDP.

for d in Data **do**
  CLIFF&MORPH$(d, r, p) \mapsto private_d$
  IPR $\leftarrow$ **IPR**$(d, private_d)$
  for d$'$ in $Data - d$ **do**
    G $\leftarrow$ **CPDP**$(d, d', learner)$
    G$'$ $\leftarrow$ **CPDP**$(private_d, d', learner)$
  end for
end for
return IPR, G, G$'$

Table 5.6: G-measures for privacy algorithms using parameter values learned from the parameter experiments from Analysis 2. These are compared with the results for the original data. The bold numbers of each row indicates that it is the highest value for the privacy algorithms.

Projects	Original	CLIFF&MORPH	Data Swapping	K-Anonymity
ant-1.7	75.1	**75.8**	61.9	66.0
camel-1.6	73.2	58.3	44.5	**59.4**
ivy-2.0	67.6	**55.1**	30.0	53.0
lucene-2.4	77.4	**71.6**	67.6	66.6
poi-3.0	71.1	52.3	**64.1**	60.4
synapse-1.2	78.5	**72.6**	57.3	58.0
velocity-1.6.1	57.7	**67.5**	63.9	43.1
xalan-2.6	69.6	**69.3**	49.2	59.3
xerces-1.3	73.5	**70.9**	63.6	60.6
Median	**73.2**	**69.3**	**61.9**	**59.4**

result for the privacy algorithms and the original data. This is according to the Mann Whitney U test [2] (P >= 0.05, two-tailed test). So the "best parameter values" work for other data.

Figure 5.6 shows the median IPR results for each privacy algorithm from applying "best" parameter values from Analysis 2. CLIFF&MORPH has the highest median IPR at **71.3** followed by

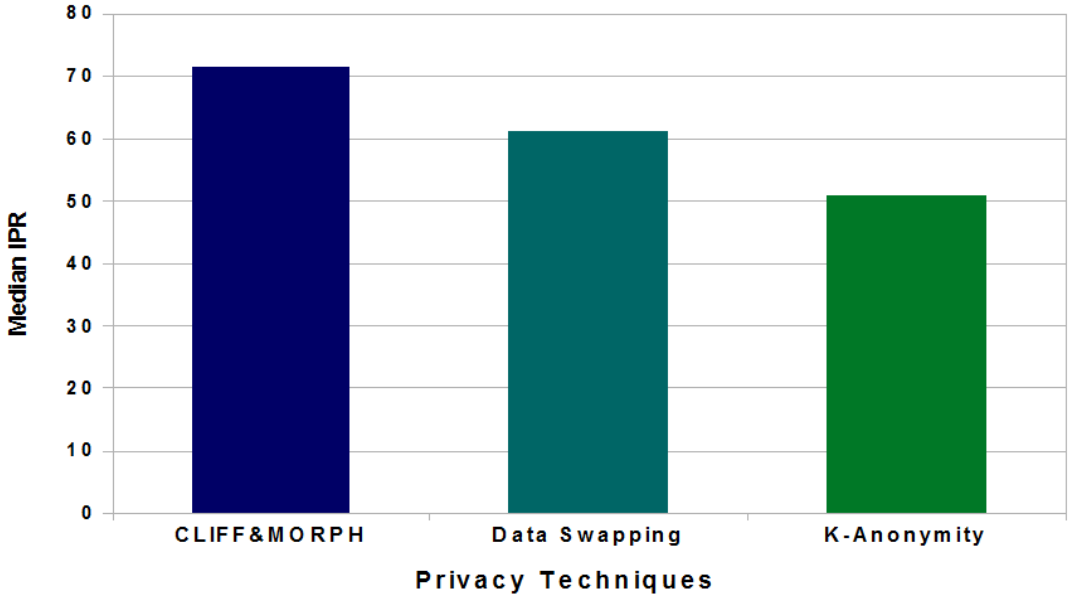


Figure 5.6: Median IPR results for each privacy algorithm from applying "best" parameter values from Analysis 2. According to the Mann Whitney U test [2] ($P < 0.05$, two-tailed test), CLIFF&MORPH has significantly better IPRs than both data swapping and *k*-anonymity.

data swapping with **61.0**, then *k*-anonymity with **50.9**. According to the Mann Whitney U test [2] ($P < 0.05$, two-tailed test), CLIFF&MORPH has significantly better IPRs than both data swapping and *k*-anonymity.

5.5.3 Discussion

From our second analysis we found that "best parameter values" for privacy algorithms can be found with a search budget of 24 runs. But, could the parameters learned be transferred to other projects to reduce the search budget? The results in Table 5.6 indicate that this maybe the case. In the best case, if the performance of transferred parameters meet the data owners' standards, the search budget is nil. In the worst case, if expectations are not met, the transferred parameters offer knowledge of what is not acceptable, thereby reducing the search budget.

Looking at Table 5.6 more closely, we see that 4 out of 9 data sets have relatively higher

g-measures for the "Original" data than the data we applied privacy algorithms to. For example, *camel-1.6* has a *g*-measure of 73.2 while the *g*-measures for the privacy algorithms range from 44.5 to 59.4. This pattern follows with *ivy-2.0, poi-3.0, and synapse-1.2*. In these cases we attribute this to information loss after applying the privacy algorithms with the learned parameters from the previous analysis. In addition, for the cases of *camel-1.6 and ivy-2.0* which both have small percentages of defects, 15% and 8.4% respectively, this severe imbalance could have contributed to the degradation from the "Original" result. We also see the opposite pattern with *velocity-1.6.1*, where its *g*-measure of 57.7 is less than that for CLIFF&MORPH and Data Swapping, but higher than that of *k*-anonymity. We attribute this result to the possible overlapping of defective instances and non-defective instances in the *velocity-1.6.1*, making it difficult for the Naive Bayes learner to distinguish between the two class values.

Using these results, we can now address RQ3 ("Are the results for parameter tuning for privacy algorithms useful for reducing the search budget?"). Our answer is as follows:

> **A3**: When it comes to both privacy (IPR) and utility (g-measure), the "best parameter values" from CLIFF&MORPH appears to be generalist in nature for the static code defect data used in this study and can reduce the search budget. In other words, we were able to successfully transfer the "best parameter values" from one train-test combination to other project combinations with significantly higher IPRs than both data swapping and *k*-anonymity. This is done with a reduced search budget since we are able to start with promising parameter vectors that produce good results.

5.6 Related Work

To to best of our knowledge, this is the first work on parameter tuning for privacy algorithms in the CPDP context. However, there are some studies on integrating background knowledge into privacy algorithms which put the burden on the data owner for setting parameters based on how

much information an attacker may have about their target. Here we use background knowledge as a means to measure privacy and found a contradictory result with swapping in that the less the attacker knew the more likely they were to discover the sensitive attribute value of their target.

Martin et al. [53] provide one of the first methods for modeling an attacker's background knowledge. Their work does not assume that you know the attacker's background knowledge, instead, they assume bounds on the attacker's knowledge in terms of the number of basic units of knowledge of the attacker. In other words the authors take the worst-case view where the attacker obtained the complete information about which individuals have records in the data set.

This work was then extended by Chen et al. [150]. They complained that the formal language developed by Martin et al. quantified background knowledge in a less than intuitive fashion. They also stated that because of this it would be difficult for the data owner to set an appropriate value for k which is the number of k implications that an attacker may know. To improve on this, Chen et al. provide an intuitive and therefore usable, quantification of an attacker's background knowledge. They consider three types of knowledge that arise naturally:

1. Knowledge about the target individual.
2. Knowledge about others.
3. Knowledge about same value families.

5.7 Conclusions

Privacy algorithms with at least one parameter value to set, places the burden on the data owner to know how the algorithm works in order the select the best parameter value(s) to produce a private data set that is less prone to malicious privacy attacks than the original data. Through a more substantial experiment than previous work [11, 12] we show that CLIFF&MORPH creates more privatized data candidates that have higher IPRs and g-measures than other privacy algorithms

studied here (Figure 5.2). Our results also indicate that setting parameters is not a trivial task. The results of a total of 192 runs show variance in the IPRs (used for measuring privacy) and/or g-measures (used for measuring utility) for the three privacy algorithms used in our study. However, we also found that the data owner does not have to embark on an exhaustive search to find the "best parameter values" to produce a private data set for publication. Last, we show that it is possible for a the "best parameter values" to generalize to other projects with results that indicate a *better* privacy algorithm than previous work.

The results of this chapter are based on a data owner preventing or reducing the risk of sensitive attribute disclosure attacks for *loc*. As shown in Section 4.3.4, IPR values can differ according to what sensitive attributes are used. In Chapter 6, in addition to focusing on private multiparty data privacy with LACE, our experiment will also look at protecting other sensitive attributes in the data.

Chapter 6

Experiment 2: LACE for Private Multiparty Data Sharing

6.1 Introduction

In Section 3.7 we recalled the result of Zimmermann et al. [19] that stated:

> *Firefox predicts IE, but IE does not predict Firefox.*

We showed that like the Firefox and IE (Internet Explorer) example, *it is possible to build predictors from open source projects for proprietary projects*. In the Zimmermann study, they did not use any transfer learning techniques to improve on the cross defect prediction result. So the follow-up question to that study is, "With the application of transfer learning techniques, can IE predict Firefox?" Without access to the IE data due to privacy concerns, we cannot answer this question for IE. As a result, in this chapter we focus on privacy for other accessible proprietary data with the scenario of private multiparty data sharing.

This is another mode of LACE where multiple data owners share privatized versions of their data, based on what others have shared. This results in a collective data set we call a *private cache*. Data owners that understand the benefits of collaboration, are wary of the consequences and/or

dangers (disclosure of sensitive data) of joining their databases. In this chapter we investigate the feasibility for allowing this collaboration while limiting the knowledge gain of another's data. In other words, if two data owners (A and B) are collaborating, *A* will not gain knowledge about *B*'s data and vice versa.

We accomplish this as follows: when each data owner has *possession* of the private cache, they observe its content and if they have different information to add, they first privatize it and then add the *new* data to the private cache. This is then passed on to the next data owner.

6.2 Experimental Setup

To evaluate LACE for the case where multiple data owners want to collaborate with each other and share their data collectively, we conduct experiments in different settings:

- cross project defect prediction without LACE and without a transfer learning technique;
- cross project defect prediction with LACE and without transfer learning with relevancy filtering;
- cross project defect prediction with LACE and with relevancy filtering [18].

Recall from Chapter 4 that to create the private cache for multiple data owners, LACE proceeds as follows:

1. Each data owner applies CLIFF to their data. Only the data selected by CLIFF are used in LACE.
2. The initiator (the first to submit data to the private cache) is chosen at random and applies LeaF (Section 4.2.4) to minimize its CLIFFed data.
3. The results are then privatized with MORPH (Section 4.2.2), and are the beginnings of the private cache.

4. The private cache is sent to the second randomly chosen site (data owner), where they test each instance of their CLIFFed data using LeaF and the private cache. The test involves each instance finding its nearest exemplar in the private cache and if their class labels are different and/or they are a certain distance away then that instance is MORPHed and added to the cache.

5. Once this is complete the private cache moves on to the next random data owner and the process repeats.

6. The protocol is complete when all data owners involved have had a chance to contribute to the private cache.

All our experiments are designed around research questions RQ4 and RQ5 from the introduction. First, to check *if LACE can provide adequate protection against sensitive attribute disclosure for each data owner* (RQ4), before adding their MORPHed exemplars to the private cache we use the new exemplars and the original data to calculate the IPR (explained in Chapter 4, Section 4.3). Second, to determine if *the data resulting from private multiparty data sharing (private cache) are useful for cross project defect prediction* (RQ5), we baseline our work with a cross project defect prediction using the original data to build the defect predictor.

For the cross project defect prediction experiments with LACE, the defect predictors are built from LACEd (exemplars in the private cache) as well as Naive Bayes [83] from WEKA [148]. We run our experiments six times and the medians for IPR and the prediction performance are calculated. This is done to avoid the order bias in randomly selecting data sets to pass the private cache. Because of the incremental learning aspect of LeaF, the order in which data owners contribute to the private cache matters. The order can affect the number of exemplars found for each data set as well as the total number of exemplars that end up in the private cache.

Table 6.1: Objective Data Sets for Open-source and Proprietary Project Data.

Defect Data	Type	# Instances	# Defects	% Defects
ant-1.7	open-source	1066	166	15.6
camel-1.6	open-source	1252	188	15.0
ivy-2.0	open-source	477	40	8.4
jEdit-4.1	open-source	644	79	12.3
lucene-2.4	open-source	536	203	37.9
poi-3.0	open-source	531	281	52.9
synapse-1.2	open-source	269	86	32.0
velocity-1.6	open-source	261	78	29.9
xalan-2.6	open-source	1170	411	35.1
xerces-1.3	open-source	545	69	12.7
prop2-ver276	proprietary	2472	334	13.5
prop5-ver185	proprietary	3260	268	8.2
prop42-ver454	proprietary	295	13	4.4

6.2.1 Data

We use a total of 13 of the static code defect data sets [15, 81] (10 open-source and 3 proprietary). Table 6.1 shows the defect data used in LACE to make the private cache in the last three gray rows. These are proprietary data sets and are chosen because of their relative similarity to the 10 open source projects in Table 6.1. The similarity of the data set are calculated using the data similarity method in Section 3.7, Figure 3.3. Figure 6.1 illustrates the results. As seen, all open source data sets are close together while the proprietary data sets are spread apart. Each of the three similar proprietary projects represents data from a data owner and are used to build the private cache. The open source projects are the test defect data used in cross defect prediction experiments with the private cache. In addition, recall that in Section 5.2.1 we presented descriptions of the open-source data shown in Table 6.1 and Table 2.1 which describes the attributes of these data.

The proprietary data (prop2-ver276, prop5-ver185, and prop42-ver454) used in this experiment are all from the insurance domain and are developed by the same software development company by international teams in a plan-driven manner [151]. Madeyski et al. [151] explain that the projects are from the insurance domain and are custom built solutions with more than five years of development history. These projects also implement different feature sets according to the individual customer requirements. Finally, all of them are developed using Java enterprise technologies and frameworks, as well as already installed in customer environments.

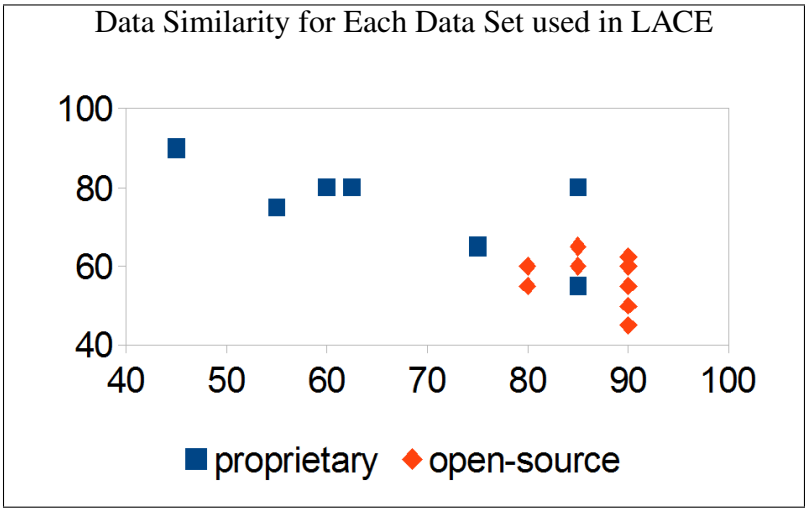


Figure 6.1: Shows the difference between the proprietary data and the open-source data.

6.2.2 Performance Evaluation

Recall that privacy algorithms have two performance objectives: 1) privacy and 2) utility. To measure the privacy offered by a privacy algorithm for a specific data set we use a Increase Privacy Ratio or IPR [12], which measures the risk of a sensitive attribute disclosure attack. We explain how IPR works in Section 4.3. In previous work IPR only measured how protected one sensitive attribute (LOC) would be if the data were published with all other attribute values changed by MORPHing and LOC values remained intact. In this chapter we investigate the IPRs when five additional sensitive attributes. In addition, we present results for when the data owner chooses to

mask the sensitive attribute values by allowing them to be MORPHed. As in Chapter 5, to measure the utility of the data generated by LACE (with sensitive attribute values masked), we measure the performance of defect predictors built with Naive Bayes [83] from WEKA [148].

Finally, recent success in cross project defect prediction is due to transfer learning. One transfer learning method is *relevancy filtering* [18, 24, 28]. Turhan et al. used a k-nearest neighbor filtering method to measure the similarity (with Euclidean distance) between test and train data. In their experiment they found *k=10* nearest training instances for each test instance. It is these selected train instances that are used in cross project defect prediction experiments. We apply Turhan's *relevancy filter* to our private cache for each test data set used in this work. However instead of using *k=10*, we tune k through initial experimentation with cross project defect prediction on the combination of the three proprietary data sets and the open source data and found that *k=5* produced the better g-measure. Therefore all our experiments use *k=5* when applying relevancy filtering.

6.3 Experimental Results

We organize our results around research questions RQ4 and RQ5 from the introduction.

6.3.1 Privacy

In this section we seek to find out if privacy offered by using LACE is adequate. Privacy is achieved in two ways: 1) Removal of data (minimization) and 2) MORPHing of the remaining data. We evaluate the privacy offered to each data owner with IPR both when sensitive attribute values intact and masked. We found that:

Table 6.2 displays the median results of IPRs of the data submitted to the private cache by each data owner (three total). Results are with the sensitive attribute values intact. The medians are calculated after six experimental runs. IPR results are shown for six individual sensitive attributes

and one combination of three sensitive attributes in the last column. This is shown for each of the three data sets that contributed to the private cache. In addition, the number and percent of exemplars contributed by each data owner are shown.

Table 6.2: This table shows the number and percentages of data added to the private cache by each data owner. Also shown are the IPRs for six sensitive attributes calculated individually then together. These results are based on the data owner sharing their data with the sensitive attribute values intact.

IPRs and the Number of Exemplars added to the Private Cache					
Sensitive Attributes	**Data**	**Size**	**# Exemplars**	**% Exemplars**	**IPR**
loc	prop42-ver454	295	5	**2%**	**94.7**
	prop2-ver276	2472	56	**2%**	**73.9**
	prop5-ver185	3260	63	**2%**	**83.3**
Total		6027	124	**2%**	**84.0**
wmc	prop42-ver454	295	15	**5%**	**76.5**
	prop2-ver276	2472	98	**4%**	**80.9**
	prop5-ver185	3260	73	**2%**	**89.3**
Total		6027	186	**3%**	**82.2**
mfa	prop42-ver454	295	9	**3%**	**57.1**
	prop2-ver276	2472	70	**3%**	**17.5**
	prop5-ver185	3260	76	**2%**	**13.0**
Total		6027	155	**3%**	**29.2**
dit	prop42-ver454	295	8	**3%**	**88.3**
	prop2-ver276	2472	67	**3%**	**73.0**
	prop5-ver185	3260	75	**3%**	**38.9**
Total		6027	150	**2%**	**66.8**
noc	prop42-ver454	295	13	**4%**	**49.6**
	prop2-ver276	2472	79	**3%**	**20.2**
	prop5-ver185	3260	87	**3%**	**11.0**
Total		6027	179	**3%**	**26.9**
rfc	prop42-ver454	295	8	**3%**	**98.2**
	prop2-ver276	2472	64	**3%**	**92.8**
	prop5-ver185	3260	58	**2%**	**95.8**
Total		6027	130	**2%**	**95.6**
loc:wmc:mfa	prop42-ver454	295	5	**2%**	**89.4**
	prop2-ver276	2472	47	**2%**	**72.8**
	prop5-ver185	3260	57	**2%**	**74.0**
Total		6027	109	**2%**	**78.7**

From the results in Table 6.2, we see that each data owner provides less than 6% of their data

to the private cache. The majority of them provide approximately 2% of their data. Table 6.2 also shows the IPR is dependent on the sensitive attributes. For five out of the seven cases of sensitive attributes studied, the average IPR is above 65%. From the remaining two (mfa and noc), the lowest average is 26.9. This is an expected result based on the IPR analysis done in Section 4.3.4. We found that these attribute values are heavily skewed and therefore an attacker's *best guess strategy* will be successful and therefore produce low IPRs. For those data sets with low IPRs for specific sensitive attributes, data owners can choose not to add their exemplars to the private cache.

Table 6.3, reports on the IPRs on the occasion where for add privacy, the data owner decides to completely mask the entire data set with CLIFF&MORPH. The table shows the IPRs for each data set participating in LACE for the different masked sensitive attributes. The Median IPRs row shows that each data set has median IPRs above 75%, while the Median Exemplars row reports the number of exemplars contributed by each data set. Finally, the Exemplars row shows that only 2% percent of the total data ended up in the private cache. Also, from the results in Table 6.3, we see that IPRs can differ according to the sensitive attributes considered by a data owner. For example, we see IPRs for 4 out of the 6 sensitive attributes measured in our experiment at 75% and above, while we have IPRs as low as 16.4% and 30.8% for "number of children" (noc) and "functional abstraction" (mfa) respectively. As explained earlier, the values of these sensitive attributes are heavily skewed and therefore it more likely for an attacker's "best-guess" to be correct. Again, in this situation a data owner can choose not to share their data.

In the following section for utility, we used the private cache generated from the data where sensitive attribute values are masked just as all the other attribute values via MORPHing.

> *Overall, LACE releases only* ***3%*** *of the 3 data sets used. Specifically, less than* ***200 out of 6027 instances*** *that are collected into the cache. Also, for the cached instances, LACE provides adequate privacy for data owners with* ***IPR above 70%*** *when the distribution of sensitive attribute values are relatively uniform (loc, wmc and rfc).*

Table 6.3: This table shows the IPRs for each data set participating in LACE for the different masked sensitive attributes. The Median IPRs row shows that each data set has median IPRs above 75%, while the Median Exemplars row reports the number of exemplars contributed by each data set. Finally, the Exemplars row shows that only 2% percent of the total data ended up in the private cache.

IPRs for Masked Sensitive Attribute Values			
Sensitive Attributes	**prop42-ver454**	**prop2-ver276**	**prop5-ver185**
loc	99.0	99.1	100.0
wmc	79.8	76.5	95.0
mfa	30.8	43.2	39.8
dit	75.0	82.8	79.8
noc	16.4	41.5	17.9
rfc	97.1	98.2	97.4
loc:wmc:mfa	75.3	64.5	77.3
Median IPRs	**75.3**	**76.5**	**79.8**
Median Exemplars	31	25	43
% Exemplars	**2%**		

6.3.2 Utility

In this section we answer the question: Are the data released by LACE useful for cross defect prediction? We compare the performance of LACE to *Original* data as well as LACE(filter) to *Original* data and measure pd, pf and *g*-measure as described in Section 5.2.4. LACE(filter) is the transfer learning element to this study that has proven to improve on CPDP results. Here we use the nearest neighbor filter from Turhan et al. [18] where each test set instance finds the k=5 nearest exemplars from the private cache. It is these selected exemplars that are used by LACE(filter) to build defect predictors. In addition, we use the Mann-Whitney statistical test at 95% confidence to determine if the result differences are statistically significant.

First Table 6.4 shows the results for CPDP with LACE compared to the original data. The first thing to notice is that there is a significant increase in the *pds* for LACE. These results comment on the benefits of sharing. LACE's intelligent selection of training data led to much higher *pds* for all ten test-data sets studied here. We attribute this better performance to LACE's careful selection of privatized data via CLIFF and LeaF. As to the *pfs*, increasing the probability of detection usually

Table 6.4: Cross project defect prediction results for open source data. Defect predictors are built from proprietary data. The classifier is Naive Bayes.

Projects	PD	PF	G-Measure
ant-1.7			
Original	39.2	5.1	55.4
LACE	**74.7**	61.2	51.1
camel-1.6			
Original	18.1	5.8	30.3
LACE	**72.9**	72.7	**39.7**
ivy-2.0			
Original	50.0	10.1	64.3
LACE	**77.5**	69.6	43.7
jEdit-4.1			
Original	48.1	4.6	64.0
LACE	**73.4**	67.6	44.9
lucene-2.4			
Original	16.7	1.5	28.6
LACE	**72.9**	63.7	**48.5**
poi-3.0			
Original	17.4	4.4	29.5
LACE	**84.3**	56.0	**57.8**
synapse-1.2			
Original	33.7	9.3	49.2
LACE	**81.4**	69.9	43.9
velocity-1.6.1			
Original	15.4	6.0	26.4
LACE	**92.3**	77.0	**36.8**
xalan-2.6			
Original	43.6	6.3	59.5
LACE	**66.7**	59.3	50.6
xerces-1.3			
Original	27.5	9.2	42.3
LACE	**59.4**	64.5	**44.4**
Average			
Original	31.0	6.2	45.0
LACE	**75.6**	66.2	**46.1**

means more false alarms (pf=66.2) as in previous work [12]. However with the added filter, we see this reduced to 33%, with pd of 51.2% still significantly higher than the original result with pd=31%.

Table 6.5: Cross project defect prediction with transfer learning in the form of relevancy filtering. As in Table 6.4, results are for open source data. Defect predictors are built from proprietary data. The classifier is Naive Bayes.

Projects	PD	PF	G-Measure
ant-1.7			
Original	39.2	5.1	55.4
LACE(filter)	**58.4**	29.7	**63.8**
camel-1.6			
Original	18.1	5.8	30.3
LACE(filter)	4.8	**1.7**	9.1
ivy-2.0			
Original	50.0	10.1	64.3
LACE(filter)	**70.0**	47.8	59.8
jEdit-4.1			
Original	48.1	4.6	64.0
LACE(filter)	**68.4**	39.8	64.0
lucene-2.4			
Original	16.7	1.5	28.6
LACE(filter)	**67.0**	49.8	**57.4**
poi-3.0			
Original	17.4	4.4	29.5
LACE(filter)	**79.4**	46.8	**63.7**
synapse-1.2			
Original	33.7	9.3	49.2
LACE(filter)	**41.9**	13.1	**56.5**
velocity-1.6.1			
Original	15.4	6.0	26.4
LACE(filter)	15.4	**2.2**	**26.6**
xalan-2.6			
Original	43.6	6.3	59.5
LACE(filter)	**64.5**	48.5	57.3
xerces-1.3			
Original	27.5	9.2	42.3
LACE(filter)	**42.0**	49.6	**45.8**
Average			
Original	31.0	6.2	45.0
LACE(filter)	**51.2**	33.0	**50.4**

Looking closer at these results we see that there are some exceptions to these observations. Specifically, in Table 6.5 where we use the filter, *camel-1.6* and *velocity-1.6.1* have either the same

or low *pds* than the *Original* data. This was not the case in Table 6.4 where no filter is used. We attribute this to the data sets being dissimilar to the proprietary data and therefore filtering this data further increased this dissimilarity thus causing the lower *pds* and *pfs*.

> *The overall results indicate that cross project defect prediction with LACEd data and relevancy filtering is* ***comparable and sometimes better*** *than defect prediction with the original data according to the average pds and g-measures.*

6.3.3 Comparison to Prior Results

First, previous work only focused only on open source data [12], here we use proprietary data for building predictors for open source data. This is the reverse of the study done by He&Peters [28].

Previous work with CLIFF&MORPH also worked best when returning 20% of data, MORPHed [12]. Here, with LeaF we are able to return approximately 3% of data with lower false alarm rates. Why is this so? We hypothesize that the LeaF algorithm and the application *relevancy filtering* are the reasons for this new better result. While LeaF seeks diversity when creating the private cache, *relevancy filtering* seeks to extract data from the private cache that are most similar to the test data.

Whatever the reason, the overall pattern is clear:

> *LACE greatly improves on our prior results by* ***halving the observed false rate from an average of 66.2% to 33%****.*

6.4 Summary

In this chapter we presented LACE as a private multiparty data sharing environment for cross project defect prediction. LACE is designed to encourage data owners to team-up to minimize and privatize their data for publication. LACE is able to produce a private cache the following qualities:

- Data in the private cache cannot be linked to the original data owner since the private cache contains subsets of MORPHed data from multiple sources.
- Depending on the distribution of the sensitive attributes, data in the private cache are private with only 3% of all data added to the private cache and IPRs above 70% when the sensitive attribute values are more uniformly distributed (Table 6.2).
- LACE's instance selection strategy is more intelligent than methods used in prior work [12]- so much so that LACE halves the observed false alarm rates.
- Data in the private cache remains useful for cross project defect prediction showing comparable results to the original data.

We hope that this result encourages more data sharing, more cross-project experiments, and more work on building software engineering models that are general to large-scale systems. We also hope one day to be able to answer the following question for ourselves,

"Can IE predict Firefox?"

Chapter 7

Threats to Validity

Feldt et al. [152] stated that a critical element of any empirical research study is to analyze and mitigate threats to the validity of the results. With any empirical study, biases can affect the final results. Therefore any conclusions made from this work must be considered with threats to validity in mind. We begin with those threats we have alleviated in our experiments and then report on the external, construct and internal validity of our work.

7.1 Alleviated Threats

We present two alleviated threats. First, in previous experiments with privacy algorithms [12], we employed a narrow range of values for the different algorithm parameters. In this dissertation, we mitigate this parameter bias by expanding on the ranges of parameter values for each privacy algorithm studied here. We therefore perform a parameter tuning experiment, running each algorithm 48 times (Chapter 5).

Second, in previous experiments [12], we measured the sensitive attribute disclosure risk of a privatized data set with IPR using only *lines of code* as the sensitive attribute. In this work, we show that IPR can be used on other and multiple sensitive attributes (Section 4.3.4 and Section 6.3.1).

Additionally, in previous work [12], this idea of IPR was to show data owners how protected their sensitive attribute data would be if released without disguise while the other attribute values were masked. Here we also add IPR results for sensitive attributes that are masked.

7.2 External Validity

External validity is concerned with whether we can generalize results outside the scope of our study. We explain these in terms of bias.

Sampling bias threatens any classification experiment; i.e., what matters there, may not be true here. For example, the data sets used here were supplied by one research group. Also, even though we used both open source and proprietary data in our studies, and the data covers a large scope of applications including text/xml processing systems, search engines, source code integration/build tools, and management information systems, they are all from Java systems. We therefore cannot assume these projects represent all projects in industry and we do not know if the results of this study generalize to other programming languages. The best we can do is define our methods and publicize our data so that other researchers can try to repeat our results and, perhaps, point out a previously unknown bias in our analysis. Hopefully, other researchers will emulate our methods in order to repeat, refute, or improve our results.

In addition, when we use the query generator described in Section 4.3.3 to model an attacker's background knowledge, we randomly create up to 1000 queries to calculate the IPR. It is therefore possible that increasing the number of queries used may change the IPR values and so affect the conclusions made in this dissertation. However we believe that the high number of queries use will mitigate against this due to the law of large numbers [153] in probability theory which claims that the more trials (queries) done, the closer the experimental result (IPR) could get to the expected value.

Learner bias: For building the defect predictors in this study, we elected to use Naive Bayes

based on its reputation for comparable performance with more complicated learners [28, 66, 69]. Classification is a large and active field and any single study can only use a small subset of the known classification algorithms. Surely other learners may produce different results, however considering the scope of our work, the "No-free-lunch" theorem [137], and the fact that the data owner has no idea how the published privatized data will be used, we determine that one indicator for utility is enough. Another issue with the learner particularly in this study of parameter tuning is that Naive Bayes also has parameters that can be adjusted based on the data, for example using a kernel estimator rather than the normal distribution for numeric attributes. To keep things simple we use the default values in the WEKA [148] implementation of Naive Bayes.

Comparison bias: There are many privacy algorithms and it would be difficult to compare the performance of CLIFF&MORPH against all of them. This dissertation compares our approach with privatization methods that are known *not* to damage classification, this is why we used the Data Swapping (also used by Taneja et al. [5]). We also used *k*-anonymity [34], a widely used privacy algorithm.

7.3 Construct Validity

Construct validity of this study mainly questions the model performance assessment approach and indicators we adopted.

Evaluation bias: As mentioned in many other related studies, the conclusion of defect prediction depends on the performance assessment indicators [20, 86]. In this study we used g-measure (Chapter 5) as well as *pd* and *pf* (Chapter 6) to present the prediction performance just as some widely cited studies did [69, 70].

Also in this dissertation we use IPR to measure the sensitive attribute disclosure risk for sensitive attributes in a data set. To calculate IPR we use background knowledge specific to the original data sets without regard for other types of background knowledge which cannot be captured by

the queries used in this study. For instance, correlation knowledge and knowledge about knowing information about related files.

Another evaluation bias involves the utility of a privatized data set. This can be measured semantically (where the workload is unknown) or empirically (known workload e.g., classification or aggregate query answering). In this work we measure utility empirically for defect prediction.

7.4 Internal Validity

With internal validity it is important to ask, "Are there alternative causes that explain my observations and results?" In our work we consider an *encoding bias*. All the privacy algorithms for this research were encoded based on the published papers in which they appeared. There is no guarantee that programming flaws or bugs were not introduced. To mitigate this threat, a code review is conducted by both the author of the code along with a lab advisor. However, since this process involves a human element, it is not full-proof and therefore the encoding bias may exist in this research.

Chapter 8

Conclusions and Future Work

Studies have shown that early detection and fixing of defects in software projects is less expensive than finding defects later on [1, 67, 68]. Organizations with local data can take full advantage of this early detection benefit by doing local defect prediction. When an organization does not have enough local data to build defect predictors, they might try to access relevant data from other organizations in order to perform cross project defect prediction. That access will be denied unless the privacy concerns of the data owners can be addressed.

In this dissertation we explored whether privacy-preserving data sharing in cross project defect prediction can be accomplished by the minimization and then constrained obfuscation of data. Our goal was to develop algorithms to privatize data and allow for data sharing while adhering to any confidentially stipulations of data. We based our research on two key insights. First, that using less data did little to degrade classification results and second, swapping data attribute values also did not greatly affect the results of classification. As a result, we began by exploring minimization and obfuscation techniques for producing privatized data candidates. Through a parameter tuning experiment, we show that our approach, CLIFF&MORPH (the privacy algorithms used in LACE) offer a better balance between privacy and utility than other state-of-the-art algorithms. Also, through private multiparty data sharing, we found that data from proprietary data can be used to

build defect predictors for open source data. Finally we found that multiple data owners who understand the benefits of sharing their data, can accomplish this by contributing just 2% of their data to a private cache.

In this concluding chapter we summarize our results in Section 8.1. In Section 8.2 we discuss broader impact goals for the presented work, future work (Section 8.3) and final remarks (Section 8.4).

8.1 Summary of Results

The summary of the results of the five research questions studied in this dissertation are shown in Table 8.1. The table informs on the questions, the sections where they are explored and answered and the key result.

8.2 Research Impacts

8.2.1 Impact on Privacy metrics

There are several privacy metrics that are either synaptic or semantic [35]. Our privacy metric, IPR is semantic in that it measures the effectiveness of privacy algorithms in protecting data from sensitive attribute disclosure attacks. IPR is independent of privacy models. In this dissertation, IPR is backed with empirical evidence for multiple sensitive attributes. IPR can serve as a decision making tool for data owners. They may have privacy policies to adhere to and they can set IPR standards for their data. So, when standards are met by a privatized data candidate, this data is publicly shared otherwise it is not shared.

Table 8.1: Summary of Research Questions

RQ#	Research Question	Section	Key Result
1	Does CLIFF&MORPH provide better balance between privacy and utility than other state-of-the-art privacy algorithms?	Section 5.3	CLIFF&MORPH provides adequate privacy compared with other privacy algorithms and utility (with comparable and better results than non-privatized data for defect prediction).
2	How hard is parameter tuning for privacy algorithms?	Section 5.4	When seeking the best parameter values for privacy algorithms, the search does not have to be exhaustive. A *best* answer can be found in a few runs, in this case 24.
3	Are the results for parameter tuning for privacy algorithms useful for reducing the search budget (multiple system runs)?	Section 5.5	We are able to successfully transfer the "best parameter values" from one train-test combination to other projects (Table 5.6).
4	Does private multiparty data sharing with LACE offer adequate privacy to data owners?	Section 6.3.1	Yes, Table 6.2 shows that overall only about 3% of all data is added the private cache for data sharing.
5	Are the data resulting from private multiparty data sharing (private cache) useful for cross project defect prediction?	Section 6.3.2	After filtering the private cache, results are comparable to the original data.

8.2.2 Impact on Cross Project Defect Prediction

Recent work in CPDP has shown success when transfer learning techniques are incorporated. This success will be for nothing if organizations cannot access to create defect predictors when they have insufficient data to build their own defect predictors. Our work on data privacy for CPDP has also added to this field of study. Through a data similarity measure (Section 3.7) we show that when projects need access to "other" data to build defect predictors, one solution is to choose data that are the most similar. This is one way to remove the brute force search that is required when trying to locate the best data for CPDP and quickly improve on transfer results.

8.2.3 Impact on Private Multiparty Data Sharing

There is much the open source community and proprietary developers can learn from each other. One way to find general trends and patterns in software engineering is to share data with others for data analysis or to replicate research studies. LACE offers a way for data owners in industry to work together to produce a private cache. One issue with data sharing among a group especially among competitors is fear of loss of competitive edge [32]. The private multiparty data sharing component of LACE allows such group sharing and allows for even less data to be shared in the group than if they applied LACE separately.

8.3 Future Work

Our experimental results have shown that it is possible to protect data from sensitive attribute disclosure, and have the data remain useful for cross project defect prediction. However it is important to continue to work on improving LACE. Our results suggest the following future work:

- The experiments in this dissertation should be repeated with other privacy algorithms and privacy measures.
- While Section 6.3.1 shows that we can increase privacy, it also shows that we cannot 100% guarantee it. At this time, we do not know the exact levels of privacy required in industry or if the results of Section 6.3.1 meet those needs. A survey of data owners to find out their privacy standards for data sharing is required.
- In the study of data privacy, modeling the attacker's background knowledge is important to determine how private a data set is. Researchers [53, 150] in this area agree that it is a challenge especially since no one knows the exact knowledge of a potential attacker. In this dissertation we only focused on background knowledge specific to the original data sets.

In other words, we try to answer the question, "What if the attacker knows this or that?" Evaluation of other types of background knowledge need to be considered for this work.

- As shown in Figure 8.1, there are other areas in software engineering that feature work in data privacy. The work on privacy and awareness [10] focuses on providing users with information to make them aware of the consequences of information disclosure. My work can relate to this by providing an extra layer of privacy for users who are willing to share their data but not their exact data i.e. if they are not happy with the potential privacy threats related to disclosure, they can choose to release an anonymous version of their data that are still useful to the receivers of the data.

  My algorithms thus far are trained on project defect data that are numerical. Adding a more social aspect to the research which puts the focus on an individual's privacy and providing them with the necessary information and control to protect that privacy is the next step.

- With Big Data Privacy, an open issue is computation cost of applying privacy algorithms prior to analysis. It is important that algorithms studied here are improved to be scalable.

- An interesting observation of our experiments is that the low dimensionality phenomenon of cross project defect prediction. This is evident in the use of approximately 3% of defect data contributed to the private cache to build defect predictors. Our IPR results (Section 6.3.1) shows that learning from less data has benefits for the privacy and security of projects that generate the training data. The low dimensionality nature of defect prediction data was first described by Menzies et al. [69]. More investigations are needed on the low dimensionality phenomenon of cross project defect prediction. The particular focus will be to try feature selection methods.

- Techniques from the field of transfer learning have been employed to improve on cross project defect prediction [18, 27, 28]. The next step is to explore cross domain knowledge

transfer, where data owners with data for different metrics such as process metrics or social network metrics, can collaborate with each other. In usual practice, the training and test sets of a data miner operate over tables of data with the same column names. From a technical perspective, that means that the training and test sets share the same *ontology*.

With transfer learning, it should be possible to relax the assumption that the source and target data has the same ontology. For example, at least in principle, a learner could learn *synonyms* between features expressed in different ontologies in different data sets. For example, given effective synonyms, it would be possible to apply lessons learned from (e.g.) *procedural systems* to *object-oriented systems*.

Note that if transfer learning can move models between data sets of different ontologies, then that would greatly increase the amount of data available for training software engineering models.

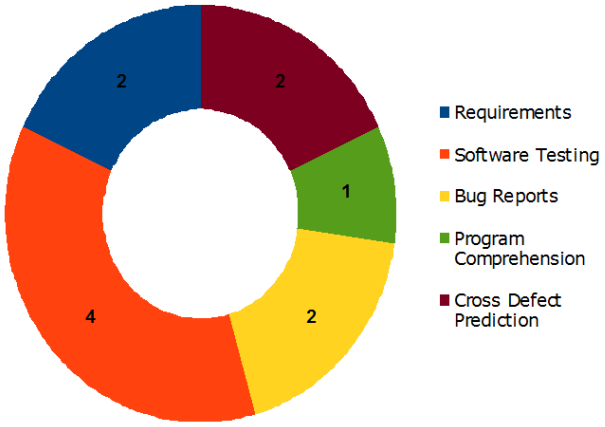


Figure 8.1: Pie chart showing privacy research in software engineering. The pie slices are sized according to the number of publications in each area of research: 1) software testing [3–6], 2) bug reporting [7, 8], 3) requirements [9, 10], 4) cross defect prediction [11, 12], and 5) program comprehension [13].

8.4 Final Remarks

The long-term goal of this work is to produce tools that will allow organizations and individuals to share data with each other with the following constraints: 1) that they are fully aware of the benefits and risks involved so they can make an informed decision about sharing, and 2) they are provided with the ability to privatized their data before release.

We believe this work was a successful step toward encouraging more data owners to share their data for research purposes, namely cross project defect prediction. Given a data set we can minimize it and privatize it while remaining useful for defect prediction. Furthermore, we provide data owners with IPR so that they will know how protected their data will be once shared. Finally, we can facilitate private multiparty data sharing particularly for those in industry willing to share but are concerned about privacy [32].

The privacy algorithms that make up LACE, (CLIFF&MORPH) and presented in this dissertation have been published at a major software engineering conference (ICSE) [11] and journal (TSE) [12]. We hope that this result encourages more data sharing in the future, more cross project experiments, and more work on building software engineering models that are general to large-scale systems.

Bibliography

[1] J. B. Dabney, G. Barber, and D. Ohi, "Predicting software defect function point ratios using a bayesian belief network," in *Proceedings of the PROMISE workshop*, 2006.

[2] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. pp. 50–60, 1947. [Online]. Available: http://www.jstor.org/stable/2236101

[3] M. Grechanik, C. Csallner, C. Fu, and Q. Xie, "Is data privacy always good for software testing?" in *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering*, ser. ISSRE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 368–377.

[4] A. Budi, D. Lo, L. Jiang, and Lucia, "kb-anonymity: a model for anonymized behaviour-preserving test and debugging data," in *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, ser. PLDI '11. New York, NY, USA: ACM, 2011, pp. 447–457. [Online]. Available: http://doi.acm.org/10.1145/1993498.1993551

[5] K. Taneja, M. Grechanik, R. Ghani, and T. Xie, "Testing software in age of data privacy: A balancing act," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser.

ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 201–211. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025143

[6] B. Li, M. Grechanik, and D. Poshyvanyk, "Sanitizing and minimizing databases for software application test outsourcing," *IEEE International Conference on Software Testing Verification and Validation*, 2014.

[7] M. Castro, M. Costa, and J.-P. Martin, "Better bug reporting with better privacy," in *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS XIII. New York, NY, USA: ACM, 2008, pp. 319–328. [Online]. Available: http://doi.acm.org/10.1145/1346281.1346322

[8] J. Clause and A. Orso, "Camouflage : Automated anonymization of field data," *Proceeding of the 33rd international conference on Software engineering*, pp. 21–30, 2011. [Online]. Available: http://portal.acm.org/citation.cfm?id=1985797

[9] T. Breaux and A. Anton, "Analyzing regulatory rules for privacy and security requirements," *Software Engineering, IEEE Transactions on*, vol. 34, no. 1, pp. 5–20, Jan 2008.

[10] I. Omoronyia, L. Cavallaro, M. Salehie, L. Pasquale, and B. Nuseibeh, "Engineering adaptive privacy: On the role of privacy awareness requirements," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 632–641. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486872

[11] F. Peters and T. Menzies, "Privacy and utility for defect prediction: Experiments with morph," in *Proceedings of the 2012 International Conference on Software Engineering*, ser. ICSE 2012. Piscataway, NJ, USA: IEEE Press, 2012, pp. 189–199. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337246

[12] F. Peters, T. Menzies, L. Gong, and H. Zhang, "Balancing privacy and utility in cross-company defect prediction," *Software Engineering, IEEE Transactions on*, vol. 39, no. 8, pp. 1054–1068, Aug 2013.

[13] M. Grechanik, C. McMillan, T. Dasgupta, D. Poshyvanyk, and M. Gethers, "Redacting sensitive information in software artifacts," 2014.

[14] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, pp. 69–81, 2010.

[15] M. Jureczko, "Significance of different software metrics in defect prediction," *Software Engineering: An International Journal*, vol. 1, no. 1, pp. 86–95, 2011.

[16] F. Shull, V. Basili, J. Carver, J. Maldonado, G. Travassos, M. Mendonca, and S. Fabbri, "Replicating software engineering experiments: addressing the tacit knowledge problem," in *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n*, 2002, pp. 7–16.

[17] B. A. Kitchenham, E. Mendes, and G. H. Travassos, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Transactions on Software Engineering*, vol. 33, pp. 316–329, 2007.

[18] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, pp. 540–578, 2009.

[19] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process." in *ESEC/SIGSOFT FSE'09*, 2009, pp. 91–100.

[20] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the "imprecision" of cross-project defect prediction," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 61:1–61:11. [Online]. Available: http://doi.acm.org/10.1145/2393596.2393669

[21] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248 – 256, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584911001996

[22] E. Kocaguneli and T. Menzies, "How to find relevant data for effort estimation?" in *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 255–264.

[23] L. L. Minku and X. Yao, "Can cross-company data improve performance in software effort estimation?" in *Proceedings of PROMISE'12*, 2012.

[24] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, vol. 19, pp. 167–199, 2012.

[25] E. Kocaguneli, G. Gay, T. Menzies, Y. Yang, and J. W. Keung, "When to use data from other projects for effort estimation," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 321–324.

[26] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann, "Local versus global lessons for defect prediction and effort estimation," *Software Engineering, IEEE Transactions on*, vol. 39, no. 6, pp. 822–834, June 2013.

[27] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *ICSE'13*. IEEE Press Piscataway, NJ, USA, 2013, pp. 802–811.

[28] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from open-source projects: An empirical study on defect prediction," in *Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on*, Oct 2013, pp. 45–54.

[29] E. Kocaguneli, B. Cukic, T. Menzies, and H. Lu, "Building a second opinion: Learning cross-company data," in *PROMSE'13*, October 2013.

[30] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology, RTI Project*, vol. 7007, no. 011, 2002.

[31] F. Rahman, S. Khatri, E. T. Barr, and P. T. Devanbu, "Comparing static bug finders and statistical prediction." in *ICSE*, 2014, pp. 424–434.

[32] S. Elbaum, A. Mclaughlin, and J. Penix, "The google dataset of testing results," june 2014. [Online]. Available: https://code.google.com/p/google-shared-dataset-of-test-suite-results

[33] L. Sweeney, "k-anonymity: A model for protecting privacy," *IEEE Security And Privacy*, vol. 10, no. 5, pp. 557–570, 2002.

[34] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 571–588, Oct. 2002.

[35] J. Brickell and V. Shmatikov, "The cost of privacy: destruction of data-mining utility in anonymized data publishing," in *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '08. New York, NY, USA: ACM, 2008, pp. 70–78.

[36] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "L-diversity: Privacy beyond k-anonymity," *ACM Trans. Knowl. Discov. Data*, vol. 1, March 2007.

[37] N. Li and T. Li, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *In Proc. of IEEE 23rd Intl Conf. on Data Engineering (ICDE 07)*, 2007.

[38] R. Hickey, "Clojure." [Online]. Available: http://clojure.org/

[39] B. Li, Y. W. Chen, and Y. Q. Chen, "The nearest neighbor algorithm of local probability centers," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 38, no. 1, pp. 141 –154, feb. 2008.

[40] D. R. Wilson and T. R. Martinez, "Reduction techniques for instance-based learning algorithms," *Machine Learning*, vol. 38, pp. 257–286, 2000.

[41] Y. Lindell and B. Pinkas, "Secure multiparty computation for privacy-preserving data mining," *Journal of Privacy and Confidentiality*, vol. 1, no. 1, p. 5, 2009.

[42] B. C. M. Fung, R. Chen, and P. S. Yu, "Privacy-Preserving Data Publishing: A Survey on Recent Developments," *Computing*, vol. V, no. 4, pp. 1–53, 2010.

[43] A. Gkoulalas-Divanis, G. Loukides, and J. Sun, "Publishing data from electronic health records while preserving privacy: A survey of algorithms," *Journal of Biomedical Informatics*, no. 0, pp. –, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1532046414001403

[44] M. Barbaro, T. Zeller, and S. Hansell, "A face is exposed for aol searcher no. 4417749," *New York Times*, vol. 9, no. 2008, p. 8, August 2006. [Online]. Available: http://www.nytimes.com/2006/08/09/technology/09aol.html

[45] T. Dalenius, "Towards a methodology for statistical disclosure control," *Statistik Tidskrift*, vol. 15, no. 429-444, pp. 2–1, 1977.

[46] S.-L. Wang, B. Parikh, and A. Jafari, "Hiding informative association rule sets," *Expert Systems with Applications*, vol. 33, no. 2, pp. 316 – 323, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741740600145X

[47] V. Verykios, E. Bertino, I. Fovin, L. Provenza, Y. Saygin, and Y. Theodoridis, "State-of-the-art in privacy preserving data mining," *SIGMOD RECORD*, vol. 33, no. 1, pp. 50–57, MAR 2004.

[48] P. Samarati and L. Sweeney, "Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression," CiteSeerX - Scientific Literature Digital Library and Search Engine [http://citeseerx.ist.psu.edu/oai2] (United States), Tech. Rep., 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.37.5829

[49] H. Park and K. Shim, "Approximate algorithms with generalizing attribute values for k-anonymity," *INFORMATION SYSTEMS*, vol. 35, no. 8, pp. 933–955, DEC 2010.

[50] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The promise repository of empirical software engineering data," June 2012. [Online]. Available: promisedata.googlecode.com

[51] D. Zhu, X.-B. Li, and S. Wu, "Identity disclosure protection: A data reconstruction approach for privacy-preserving data mining," *DECISION SUPPORT SYSTEMS*, vol. 48, no. 1, Sp. Iss. SI, pp. 133–140, DEC 2009.

[52] C. C. Aggarwal, "On k-anonymity and the curse of dimensionality," in *Proceedings of the 31st International Conference on Very Large Data Bases*, ser. VLDB '05. VLDB Endowment, 2005, pp. 901–909. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083592.1083696

[53] D. Martin, D. Kifer, A. Machanavajjhala, J. Gehrke, and J. Halpern, "Worst-case background knowledge for privacy-preserving data publishing," in *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 2007, pp. 126–135.

[54] N. Zhang and W. Zhao, "Privacy-preserving data mining systems," *Computer*, vol. 40, no. 4, pp. 52–58, April 2007.

[55] C. Giannella, K. Liu, and H. Kargupta, "On the privacy of euclidean distance preserving data perturbation," *CoRR*, vol. abs/0911.2942, 2009.

[56] I. Dinur and K. Nissim, "Revealing information while preserving privacy," in *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2003, pp. 202–210.

[57] C. Dwork, "Differential privacy," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, Eds. Springer Berlin Heidelberg, 2006, vol. 4052, pp. 1–12.

[58] C. Dwork, "Differential privacy: A survey of results," *Theory and Applications of Models of Computation*, pp. 1–19, 2008.

[59] A. Chin and A. Klinefelter, "Differential privacy as a response to the reidentification threat: The facebook advertiser case study," *North Carolina Law Review*, vol. 90, no. 5, 2012.

[60] L. Sweeney, "Datafly: A system for providing anonymity in medical data," in *Proceedings of the IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI: Status and Prospects*. London, UK, UK: Chapman & Hall, Ltd., 1998, pp. 356–381. [Online]. Available: http://dl.acm.org/citation.cfm?id=646115.679937

[61] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Incognito: Efficient full-domain k-anonymity," in *Proceedings of the 2005 ACM SIGMOD International Conference on*

Management of Data, ser. SIGMOD '05. New York, NY, USA: ACM, 2005, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066164

[62] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati, "Microdata protection," in *Secure Data Management in Decentralized Systems*, ser. Advances in Information Security, T. Yu and S. Jajodia, Eds. Springer US, 2007, vol. 33, pp. 291–321. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-27696-0_9

[63] Q. Zhang, N. Koudas, D. Srivastava, and T. Yu, "Aggregate Query Answering on Anonymized Tables," *2007 IEEE 23rd International Conference on Data Engineering*, pp. 116–125, 2007.

[64] V. Torra, Y. Endo, and S. Miyamoto, "On the comparison of some fuzzy clustering methods for privacy preserving data mining: towards the development of specific information loss measures," *Kybernetika*, vol. 45, no. 3, pp. 548–560, 2009.

[65] Y. Rachlin, K. Probst, and R. Ghani, "Maximizing privacy under data distortion constraints in noise perturbation methods," in *Privacy, Security, and Trust in KDD*, ser. Lecture Notes in Computer Science, F. Bonchi, E. Ferrari, W. Jiang, and B. Malin, Eds. Springer Berlin Heidelberg, 2009, vol. 5456, pp. 92–110. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01718-6_7

[66] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *Software Engineering, IEEE Transactions on*, vol. 34, no. 4, pp. 485 –496, july-aug. 2008.

[67] B. Boehm and P. Papaccio, "Understanding and controlling software costs," *IEEE Trans. on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, October 1988.

[68] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz, "What we have learned about fighting defects," in *Proceed-*

ings of the 8th International Symposium on Software Metrics, ser. METRICS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 249–258.

[69] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *Software Engineering, IEEE Transactions on*, vol. 33, no. 1, pp. 2 –13, jan. 2007.

[70] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10515-010-0069-5

[71] T. McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, vol. SE-2, no. 4, pp. 308–320, Dec 1976.

[72] R. Vasa, "Growth and change dynamics in open source software systems," Ph.D. dissertation, Faculty of Information and Communication Technologies,Swinburne University of Technology, 2010.

[73] M. Lumpe, S. Mahmud, and R. Vasa, "On the use of properties in java applications," in *Software Engineering Conference (ASWEC), 2010 21st Australian*, April 2010, pp. 235–244.

[74] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *Software Engineering, IEEE Transactions on*, vol. 20, no. 6, pp. 476–493, Jun 1994.

[75] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 580–586. [Online]. Available: http://doi.acm.org/10.1145/1062455.1062558

[76] J. Turner, "A predictive approach to eliminating errors in software code," http://spinoff.nasa.gov/Spinoff2006/ct_1.html, 2006, accessed: 2014-07-10.

[77] A. Tosun, B. Turhan, and A. Bener, "Practical considerations in deploying ai for defect prediction: a case study within the turkish telecommunication industry," in *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, ser. PROMISE '09. New York, NY, USA: ACM, 2009, pp. 11:1–11:9.

[78] A. T. Misirli, A. B. Bener, and R. Kale, "Ai-based software defect predictors: Applications and benefits in a case study." *AI Magazine*, vol. 32, no. 2, pp. 57–68, 2011.

[79] S. J. Pan and Q. Yang, "A survey on transfer learning," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 10, pp. 1345–1359, Oct 2010.

[80] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," in *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, ser. ISSTA '04. New York, NY, USA: ACM, 2004, pp. 86–96. [Online]. Available: http://doi.acm.org/10.1145/1007512.1007524

[81] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '10. New York, NY, USA: ACM, 2010, pp. 9:1–9:10. [Online]. Available: http://doi.acm.org/10.1145/1868328.1868342

[82] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[83] D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," in *Machine Learning: ECML-98*, ser. Lecture Notes in Computer Science, C. Ndellec and C. Rouveirol, Eds. Springer Berlin / Heidelberg, 1998, vol. 1398, pp. 4–15.

[84] W. Afzal, "Using faults-slip-through metric as a predictor of fault-proneness," in *Proceedings of the 2010 Asia Pacific Software Engineering Conference*, ser. APSEC '10, 2010, pp. 414–422.

[85] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," in *Proceedings of the 4th international workshop on Predictor models in software engineering*, ser. PROMISE '08. New York, NY, USA: ACM, 2008, pp. 47–54.

[86] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, vol. 13, pp. 561–595, 2008.

[87] S. Hido, T. Idé, H. Kashima, H. Kubo, and H. Matsuzawa, "Unsupervised change analysis using supervised learning," in *Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining*, ser. PAKDD'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 148–159.

[88] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[89] N. E. Fenton and M. Neil, "Software metrics: successes, failures and new directions," *Journal of Systems and Software*, vol. 47, no. 2-3, pp. 149 – 157, 1999.

[90] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley, 2012. [Online]. Available: http://books.google.com/books?id=Br33IRC3PkQC

[91] J. Vaidya and C. Clifton, "Privacy-preserving data mining: why, how, and when," *Security Privacy, IEEE*, vol. 2, no. 6, pp. 19–27, Nov 2004.

[92] T. Cover and P. Hart, "Nearest neighbor pattern classification," *Information Theory, IEEE Transactions on*, vol. 13, no. 1, pp. 21–27, Jan 1967.

[93] G. Gates, "The reduced nearest neighbor rule (corresp.)," *Information Theory, IEEE Transactions on*, vol. 18, no. 3, pp. 431 – 433, 1972.

[94] P. Hart, "The condensed nearest neighbor rule (corresp.)," *Information Theory, IEEE Transactions on*, vol. 14, no. 3, pp. 515 – 516, may 1968.

[95] B. Dasarathy, "Minimal consistent set (mcs) identification for optimal nearest neighbor decision systems design," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 24, no. 3, pp. 511–517, Mar 1994.

[96] J. A. Olvera-López, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad, "A new fast prototype selection method based on clustering," *Pattern Analysis and Applications*, vol. 13, no. 2, pp. 131–141, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10044-008-0142-x

[97] D. Aha, D. Kibler, and M. Albert, "Instance-Based Learning Algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, JAN 1991.

[98] J. Bezdek and L. Kuncheva, "Nearest prototype classifier designs: An experimental study," *International Journal of Intelligent Systems*, vol. 16, no. 12, pp. 1445–1473, DEC 2001.

[99] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data Mining and Knowledge Discovery*, vol. 6, no. 2, pp. 153–172, APR 2002.

[100] J. Cano, F. Herrera, and M. Lozano, "Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 6, pp. 561–575, DEC 2003.

[101] C.-H. Chou, B.-H. Kuo, and F. Chang, "The generalized condensed nearest neighbor rule as a data reduction method." *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, pp. 556 – 559, sep 2006.

[102] S. García, J. R. Cano, and F. Herrera, "A memetic algorithm for evolutionary prototype selection: A scaling up approach," *Pattern Recognition*, vol. 41, no. 8, pp. 2693–2709, 2008.

[103] A. Lumini and L. Nanni, "A clustering method for automatic biometric template selection," *Pattern Recognition*, vol. 39, no. 3, pp. 495–497, MAR 2006.

[104] B. Narayan, C. Murthy, and S. Pal, "Maxdiff kd-trees for data condensation," *Pattern Recognition Letters*, vol. 27, no. 3, pp. 187–200, FEB 2006.

[105] J. Arturo Olvera-López, J. Ariel Carrasco-Ochoa, and J. Francisco Martínez-Trinidad, "Object selection based on clustering and border objects," in *Computer Recognition Systems 2*, ser. Advances in Intelligent and Soft Computing, M. Kurzynski, E. Puchala, M. Wozniak, and A. Zolnierek, Eds. Springer Berlin / Heidelberg, 2007, vol. 45, pp. 27–34.

[106] J. A. Olvera-López, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad, "Prototype selection via prototype relevance," in *Progress in Pattern Recognition, Image Analysis and Applications*, ser. Lecture Notes in Computer Science, J. Ruiz-Shulcloper and W. Kropatsch, Eds. Springer Berlin / Heidelberg, 2008, vol. 5197, pp. 153–160.

[107] T. Raicharoen and C. Lursinsap, "A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (POC-NN) algorithm," *Pattern Recognition Letters*, vol. 26, no. 10, pp. 1554–1567, JUL 15 2005.

[108] J. Riquelme, J. Aguilar-Ruiz, and M. Toro, "Finding representative patterns with ordered projections," *Pattern Recognition*, vol. 36, no. 4, pp. 1009–1018, APR 2003.

[109] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour, "An algorithm for a selective nearest neighbor decision rule (corresp.)," *Information Theory, IEEE Transactions on*, vol. 21, no. 6, pp. 665 – 669, nov 1975.

[110] A. Srisawat, T. Phienthrakul, and B. Kijsirikul, "Sv-knnc: An algorithm for improving the efficiency of k-nearest neighbor," in *PRICAI 2006: Trends in Artificial Intelligence*, ser. Lecture Notes in Computer Science, Q. Yang and G. Webb, Eds. Springer Berlin Heidelberg, 2006, vol. 4099, pp. 975–979. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-36668-3_117

[111] I. TOMEK, "Experiment with Edited Nearest-Neighbor Rule," *IEEE TRANSACTIONS ON SYSTEMS MAN AND CYBERNETICS*, vol. 6, no. 6, pp. 448–452, 1976.

[112] C. Veenman and M. Reinders, "The nearest subclass classifier: A compromise between the nearest mean and nearest neighbor classifier," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 9, pp. 1417–1429, SEP 2005.

[113] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 2, no. 3, pp. 408 – 421, jul 1972.

[114] J. A. Olvera-López, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. Kittler, "A review of instance selection methods," *Artificial Intelligence Review*, vol. 34, no. 2, pp. 133–143, 2010.

[115] V. S. Devi and M. N. Murty, "An incremental prototype set building technique," *Pattern Recognition*, vol. 35, no. 2, pp. 505 – 513, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/B6V14-44HT45G-K/2/a27d8d6d4a216b97974cfa5cd7947419

[116] Y. Li, M. Xie, and T. Goh, "A study of project selection and feature weighting for analogy based software cost estimation," *Journal of Systems and Software*, vol. 82, pp. 241–252, 2009.

[117] J. Bezdek, T. Reichherzer, G. Lim, and Y. Attikiouzel, "Multiple-prototype classifier design," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 28, no. 1, pp. 67–79, Feb 1998.

[118] J. R. Cano, F. Herrera, and M. Lozano, "Stratification for scaling up evolutionary prototype selection," *Pattern Recognition Letters*, vol. 26, no. 7, pp. 953 – 963, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/B6V15-4DTTHDM-1/2/1c98d61e651bcd792d23fce1360b91c9

[119] U. Garain, "Prototype reduction using an artificial immune model," *Pattern Anal. Appl.*, vol. 11, no. 3-4, pp. 353–363, 2008.

[120] F. Peters, "Cliff: Finding prototypes for nearest neighbor algorithms with application to forensic trace evidence," Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2010, copyright ProQuest, UMI Dissertations Publishing 2010. [Online]. Available: http://search.proquest.com/docview/859578571?accountid=2837

[121] O. Jalali, T. Menzies, and M. Feather, "Optimizing requirements decisions with keys," in *Proceedings of the PROMISE 2008 Workshop (ICSE)*, 2008.

[122] S. Kotsiantis and D. Kanellopoulos, "Discretization techniques: A recent survey," *GESTS International Transactions on Computer Science and Engineering*, vol. 32, no. 1, pp. 47–58, 2006.

[123] C. Faloutsos and K.-I. Lin, "Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," *SIGMOD Rec.*, vol. 24, no. 2, pp. 163–174, May 1995. [Online]. Available: http://doi.acm.org/10.1145/568271.223812

[124] S. K. Smit and A. Eiben, "Comparing parameter tuning methods for evolutionary algorithms," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, 2009, pp. 399–406.

[125] S. Smit and A. Eiben, "Parameter tuning of evolutionary algorithms: Generalist vs. specialist," in *Applications of Evolutionary Computation*, ser. Lecture Notes in Computer Science, C. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekrt, A. Esparcia-Alcazar, C.-K. Goh, J. Merelo, F. Neri, M. Preu, J. Togelius, and G. N. Yannakakis, Eds. Springer Berlin Heidelberg, 2010, vol. 6024, pp. 542–551.

[126] A. Eiben and S. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 19 – 31, 2011.

[127] M. Harman and B. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, pp. 833–839, December 2001.

[128] M. Harman, "The current state and future of search based software engineering," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 342–357. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.29

[129] T. Menzies, O. Elrawas, J. Hihn, M. Feathear, B. Boehm, and R. Madachy, "The business case for automated software engineerng," in *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. New York, NY, USA: ACM, 2007, pp. 303–312.

[130] Y. Yang and G. Webb, "Weighted proportional k-interval discretization for naive-bayes classifiers," in *Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2003)*, 2003.

[131] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, October 2002.

[132] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, "On the use of relevance feedback in ir-based concept location," in *IEEE ICSM'09*, 2009.

[133] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Trans. Software Eng*, vol. 32, no. 1, pp. 4–19, 2006. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TSE.2006.3

[134] A. Marcus, J. I. Maletic, and A. Sergeyev, "Recovery of traceability links between software documentation and source code," *International Journal of Software Engineering and Knowledge Engineering*, pp. 811–836, 2005.

[135] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," *Software Engineering, IEEE Transactions on*, vol. 33, no. 6, pp. 420–432, 2007.

[136] A. Arcuri and G. Fraser, "On parameter tuning in search based software engineering," in *Search Based Software Engineering*, ser. Lecture Notes in Computer Science, M. Cohen and M. Cinnide, Eds. Springer Berlin Heidelberg, 2011, vol. 6956, pp. 33–47.

[137] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 67–82, 1997.

[138] C. MacNeill, "Apache ant," may 2014. [Online]. Available: http://ant.apache.org/

[139] C. Ibsen and J. Anstey, *Camel in action*. Manning Publications Co., 2010.

[140] "Apache ivy." [Online]. Available: http://ant.apache.org/ivy/

[141] M. Jureczko, "Open source project descriptions," 2011. [Online]. Available: http://madeyski.e-informatyka.pl/download/JureczkoMadeyskiOpenSourceProjects.pdf

[142] "Apache lucene - welcome to apache lucene." [Online]. Available: http://lucene.apache.org/

[143] "Apache poi - the java api for microsoft documents." [Online]. Available: http://poi.apache.org/

[144] "Apache synapse - the lightweight esb," 2012. [Online]. Available: http://synapse.apache.org/

[145] "Apache velocity site - the apache velocity project," nov 2010. [Online]. Available: http://velocity.apache.org/

[146] "Xalan-java version 2.7.1." [Online]. Available: http://xml.apache.org/xalan-j/

[147] "Xerces java parser readme." [Online]. Available: http://xerces.apache.org/xerces-j/

[148] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, pp. 10–18, November 2009.

[149] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'"," *IEEE Trans. Softw. Eng.*, vol. 33, no. 9, pp. 637–640, Sep. 2007.

[150] B.-C. Chen, K. Lefevre, and R. Ramakrishnan, "Privacy Skyline : Privacy with Multidimensional Adversarial Knowledge," in *Organization*, ser. VLDB '07. VLDB Endowment, 2007, pp. 770–781.

[151] L. Madeyski and M. Jureczko, "Which process metrics can significantly improve defect prediction models? an empirical study," 2014. [Online]. Available: http://madeyski.e-informatyka.pl/download/Madeyski14SQJ.pdf

[152] R. Feldt and A. Magizinius, “Validity threats in empirical software engineering research - an initial survey,” in *Proc. of the Int’l Conf. on Software Engineering and Knowledge Engineering*, 2010, pp. 374–379.

[153] E. Gregersen, *The Britannica Guide to Statistics and Probability*, ser. Britannica guide series. Britannica Educational Pub., 2010. [Online]. Available: http://books.google.ie/books?id=CI111lrB6vsC