Sample Size vs. Bias in Defect Prediction

Foyzur Rahman
Univ. of California, Davis, CA
USA
mfrahman@ucdavis.edu

Daryl Posnett
Univ. of California, Davis, CA
USA
dpposnett@ucdavis.edu

Israel Herraiz
Universidad Politécnica de
Madrid, Spain
israel.herraiz@upm.es

Premkumar Devanbu
Univ. of California, Davis, CA
USA
ptdevanbu@ucdavis.edu

ABSTRACT

Most empirical disciplines promote the reuse and sharing of datasets, as it leads to greater possibility of replication. While this is increasingly the case in Empirical Software Engineering, some of the most popular bug-fix datasets are now known to be *biased.* This raises two significant concerns: first, that sample bias may lead to underperforming prediction models, and second, that the external validity of the studies based on biased datasets may be suspect. This issue has raised considerable consternation in the ESE literature in recent years. However, there is a confounding factor of these datasets that has not been examined carefully: size. Biased datasets are sampling only *some* of the data that could be sampled, and doing so in a biased fashion; but biased samples could be smaller, or larger. Smaller data sets in general provide less reliable bases for estimating models, and thus could lead to inferior model performance. In this setting, we ask the question, *what affects performance more, bias, or size?* We conduct a detailed, large-scale meta-analysis, using simulated datasets sampled with bias from a high-quality dataset which is relatively free of bias. Our results suggest that size always matters just as much bias direction, and in fact *much more* than bias direction when considering information-retrieval measures such as AUC and F-SCORE. This indicates that at least for prediction models, even when dealing with sampling bias, simply finding larger samples can sometimes be sufficient. Our analysis also exposes the complexity of the bias issue, and raises further issues to be explored in the future.

Categories and Subject Descriptors

K.6.3 [**Software Management**]: Software maintenance

General Terms

Management

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ESEC/FSE '13, August 18–26, 2013, Saint Petersburg, Russia
Copyright 13 ACM 978-1-4503-2237-9/13/08 ...$15.00.

Keywords

Bias, size, defect prediction

1. INTRODUCTION

Detailed data on bugs are clearly crucial to empirical studies of software quality. Such data is generally collected in *bug-fix datasets.* Several such datasets have been publicy released into repositories such as PROMISE[1]. These datasets include both detail on reported bugs, and the source code location where the bugs were fixed. The fix location is provided by a *link* to commits in the version control system. These links identify the source code files involved in a bug report, as well as other details, such as the developer who committed the fix, the date and time, and the lines changed in the corresponding files. This is a rich source of historical data for building software defect prediction models that may yield improved understanding of the factors that affect software quality.

These links, between bugs and commits, are recovered through a *post facto* examination of the commit logs in version control, and/or the comments and meta-data associated to the bug report; both of these are *manually entered by code contributors.* Thus, the ability to recover these links depend on the tagging practices of the contributors to the software project under study. These practices vary across each project; this heterogeneity makes the consistent, reliable recovery of all of the links between bugs and commits problematic. Indeed, many datasets widely used within the research community are missing some links and have been found to be incomplete and biased [2, 3].

Training defect prediction models using biased and/or incomplete datasets is problematic. Wu *et al.* [21] compared the accuracy of a prediction model trained using three different datasets: a biased dataset obtained using heuristics, a putatively unbiased dataset obtained manually and a putatively biased dataset obtained using an automated method. The model trained with the biased dataset did not perform as well as the models trained with unbiased datasets.

This is a disappointing result, suggesting that, despite all the hard work invested in creating them, one should abandon datasets that may be biased. In this paper, we therefore take a contrarian view on bias. *Available datasets could possibly be biased, but still be large in size. Can such datasets still be useful in some prediction settings?* If it should turn out

[1]`http://promisedata.googlecode.com`

that obtaining larger training datasets (of bugs with linked fixing commits) is more important than the difficult task of obtaining entirely unbiased datasets, then this would be good news: large datasets could be used to train prediction models, with some confidence that such prediction models would still perform usefully.

In this research, *our goal is to do a controlled study of the relative effects of bias and sample size on defect prediction.* In order to do this, we begin with a very high-quality dataset, wherein linkage rates are very high (median of 80%, as compared to the more typical 50% [4] for other datasets). We then artificially sample sub-datasets with varying levels of size and bias, by selectively sampling linked defects based on biases that are known or suspected to exist. Using these artificially biased sub-datasets, we examine the relative effects of bias and sample size on defect prediction models. We use a *meta-modeling* approach, constructing *meta-models that explain the performance of prediction models.* Our research makes the following contributions.

- We examine the effects of bias on defect prediction, by artificially sampling with bias from a dataset known to be very high quality.
- We study the relative effects of different types of bias on defect prediction, using meta-models to analyze the prediction models.
- We use the meta-models to study the relative effects of bias and sample size on defect prediction.
- Finally we introduce and examine the effects of *pollution* where data sets that fail to link defects that were actually fixed introduce *false negatives*, e.g. files that are really defective but are not marked as such.

2. RELATED WORK AND THEORY

Defect prediction models use supervised methods to learn the association of different predictors and the defect proneness of entities. A labeled training dataset labels each entity as defective or not. Existing research identifies defective entities *post facto*, based on the locations where defect repair occurs. This requires clear identification of a defect-fixing activity.

Past research has used developers' comments associated with the source code changes to determine whether a change is defect-fixing. Ideally, a responsible developer always identifies defect fixes in the change log. Keywords such as "bug", "fixed" etc. in the change log message could be detected by a tool to identify defect-fixing changes [14]. One can also identify numerical bug-ids mentioned in the change log and match those ids with defect database such as Bugzilla to locate defect-fixing changes [7, 8]. However, developers do not always annotate a change with proper description; this impairs the automated identification of defect-fixing changes. Bird *et al.* [4] found that automated process only identified less than 50% of the defect fixing changes in most cases. Bachmann *et al.* [2] noted that *missing links*, the defect-fixing changes that automated process fails to recover, may impact the prediction performance of supervised learners. .

Bias Missing links can confuse and hinder a supervised learner. Links may be systematically missing, based on the properties of a defect-fixing change such as, *e.g.*, severity of the defect or the experience of the fixer. The resulting biased dataset may trick the supervised learner. For example, suppose only experienced developers are annotating their changes. Automated tools will only identify defect-fixing changes made by experienced developers; we will have over representation of the entities fixed by experienced developers. In earlier work [4] we introduced the notion of bias, and have provided formal probabilistic definitions of bias. For our purposes, here, it is sufficient to informally define **bias** as the situation where *the linked bug sample distributions of the co-variates of interest are systematically different from the distributions of the same co-variates among the entire population of fixed bugs.* Consequently, the linked sample is not truly representative of the population. So for example, the distribution of developer experience among the *linked*, fixed bugs would differ from the distribution of developer experience among *all* of the fixed bugs.

Different properties of defect-fixing changes may introduce different types of bias. Thus, as discussed above, developer experience may introduce EXPERIENCE bias. Similarly, severity of the defects may introduce SEVERITY bias. Bird *et al.* [4] found that defect-fixing changes of less severe defects are more likely to be linked. However, given a dataset, we may not know the source of bias, *e.g.*, whether it is EXPERIENCE or SEVERITY. If different sources of BIAS have different levels of influence on prediction performance, BIAS would be more damaging due to the uncertainty over the source of bias. We therefore study whether different sources of BIAS have different impacts on prediction performance. We note here that Kim *et al.* [10] have studied the influence of *noise* on bug prediction. They also propose an algorithm to find noisy instances in bug datasets, so they can be removed to avoid potential problems. Our concern is more with systematic bias, rather than noise.

Research Question 1: Do different sources of bias have varying impact on prediction performance?

Pollution The second side-effect of missing links, which to our knowledge has not been addressed, is *false negatives.* An unlinked defect-fixing change fails to identify the defective files associated with that change. This also effectively labels files that are actually defective as defect free! These false-negatives constitute a form of pollution that might affect supervised learners. We informally define **pollution** as *an erroneous condition of bug-fix datasets where files that are, in fact, defective are incorrectly labeled as defect-free.*

Besides these two consequences, missing links, inherently reduce the number of defect-fixing changes available. We use the term SIZE to represent the total number of links available. Increasing SIZE may increase the performance of the learner and may also ameliorate the impact of BIAS and POLLUTION. SIZE is also a more tractable problem; larger datasets may be easier to obtain than data sets that are both unbiased and unpolluted.

While existing research studies BIAS and the presence of missing links, so far, none have compared the effects of SIZE, POLLUTION and BIAS, to determine the relative effects of each on the performance of prediction models; this motivates our core research question.

Table 1: Studied Projects and Release Information

Project	Description	Releases	Avg Files	Avg SLOC	Link Rate
CXF	Services Framework	July 2007–April 2012, 6 releases	4038.33	358846.67	0.77
Camel	Enterprise Integration Framework	January 2008–March 2012, 8 releases	4600.38	241668.12	0.84
Derby	Relational Database	February 2006–October 2011, 7 releases	2497.29	530633.00	0.85
Felix	OSGi R4 Implementation	August 2007–November 2011, 9 releases	2740.56	249886.22	0.85
HBase	Distributed Scalable Data Store	Jun 2007–May 2012, 8 releases	934.75	187953.38	0.85
HadoopC	Common libraries for Hadoop	Jun 2007–September 2009, 6 releases	1047.17	142257.33	0.79
Hive	Data Warehouse System for Hadoop	October 2008–May 2012, 7 releases	966.29	152079.86	0.75
Lucene	Text Search Engine Library	October 2005–November 2009, 7 releases	990.86	122527.00	0.85
OpenEJB	Enterprise Java Beans	August 2007–April 2012, 7 releases	2895.43	225018.43	0.86
OpenJPA	Java Persistence Framework	January 2007–February 2012, 8 releases	3181.50	321033.50	0.92
Qpid	Enterprise Messaging system	November 2008–May 2012, 7 releases	1724.00	198311.86	0.73
Wicket	Web Application Framework	November 2008–May 2012, 5 releases	2295.20	152565.40	0.77

Table 2: Process Metrics

Short Name	Description
COMM	Commit Count
ADEV	Active Dev Count
DDEV	Distinct Dev Count
ADD	Normalized Lines Added
DEL	Normalized Lines Deleted
OWN	Owner's Contributed Lines
MINOR	Minor Contributor Count
NADEV	Neighbor's Active Dev Count
NDDEV	Neighbor's Distinct Dev Count
NCOMM	Neighbor's Commit Count
OEXP	Owner's Experience
EXP	All Committer's Experience

Research Question 2: Considering BIAS, POLLUTION, and SIZE, which aspect of missing links affects prediction models the most?

False Positives Finally, it is also possible that the bug links contain *false positives*, *viz.*, edits that are not really bug fixes, but accidentally get included into bug-fixing commit. This issue is outside the scope of our current research, and we hope to address this in the future. However, since developers may just accidentally include non-bug-fixing changes in a bug-fixing commit, our belief is that this type of data pollution is more likely to be characteristically *noisy* rather than biased. Arguably, the approach introduced by Kim *et al.* [10] for dealing with noise should be effective in dealing with false positives.

3. EXPERIMENTAL METHODOLOGY

Projects Studied Table 1 shows the 12 open source projects studied in this paper. In addition to release information and project size, the table also lists the median percentage of defects for which we could identify the fixing-commits (link rates). All are Apache Software Foundations projects, written in Java; however, they come from a very diverse range of domains. For each project we downloaded the GIT repository [2] and extracted the full commit history. We also used GIT BLAME on every file at every release to identify the contributors' information with more detail. We carefully ignored whitespace changes and code movement during our BLAME process to identify the correct provenance of each line.

All the projects shown in Table 1 use JIRA[3] issue tracking system. We mined JIRA to extract all the data associated to each bug report. Moreover, thanks to the linking and tagging practices of the ASF projects, as well as to the features of JIRA bug tracking system, we were able to retrieve the related commits which fixed each bug. We only considered Jira entries clearly identified as defects; we ignored feature changes, refactorings, etc. We then locate those fixing commits in GIT to extract commit information, such as changed lines and commit author. Any files modified in these bug-fixing commits are considered as defective *post-facto*.

Jira Data All of our projects have *very high linking rates*, with a median rate of approximately 80%. Even at these high link rates, our data indicates a slight bias; *e.g.*, severe defects are linked at a median rate of about 80% and less severe defects at about 75%. Nevertheless, these linking rates are much higher than the 50% rates reported in prior papers on bias [2, 4]. From these very highly linked bug-fix data sets, we deliberately choose samples with higher and *and* lower bias, to study the effects of bias.

Predicting Defects Following common research practice, we study prediction at the file level. We use Logistic Regression from the WEKA toolkit to compute a probability that a file will be defective or not in a subsequent release[4]. The models are trained in a prediction setting, *viz.*, we train the model on k-th release, we test the model in $k+1$-th release, using process attributes with a binary response indicating whether a file is defective. We describe our process metrics in detail below.

Since we are interested in obtaining the best prediction model possible, we want to use as many variables as we can to capture as much variation as possible. Multicollinearity, *viz.*, strong correlation of two or more predictor variables, can be an issue with models with many variables. Multicollinearity is typically mitigated through the use of either a manual or automated stepwise procedure where a discrete subset

[2]http://git.apache.org
[3]https://issues.apache.org/jira/
[4]http://www.cs.waikato.ac.nz/ml/weka/

of the variables are selected for the model. Because we are evaluating the use of biased samples we build a very large number of prediction models; consequently, it is impractical to individually determine a particular set of predictors to use in each prediction model manually, and an automated stepwise procedure would dramatically increase our runtime. As an alternative, we use *ridge regression* [12].

Ridge regression introduces (coefficient) bias, reducing variance in the coefficients, while improving the stability of the model. The technical details are beyond the scope of this paper[5]. In the best case, the variance reduction is significantly larger in magnitude than the introduced bias. Because we are building prediction models, however, this is not a significant concern as we are not interpreting the (prediction) regression coefficients. Since we are introducing (sampling) bias into our training sets, which may disturb the relationship between the training set and test set distributions, ridge regression provides insurance that we can be reasonably certain that the (coefficient) bias introduced by multicollinearity is not impacting the quality of our prediction models in the face of (sample) bias introduced by our experimental setup.

Our choice to use Logistic regression is motivated by its popularity in empirical software engineering research. Moreover, researchers have found that the choice of metrics, rather than classification methods, is the primary driver of prediction performance [1].

Predictor Metrics Table 2 shows the process metrics we use in the prediction models.

- *COMM* measures the number of commits made to a file during a release.
- *ADEV* is the number of developers who made changes to that file during a release.
- *DDEV* is the number of distinct developers contributing to this file up to this release.
- *ADD* and *DEL* are the normalized (by the total number of added and deleted lines) added and deleted lines in the file during a release.
- *OWN* is the percentage of the lines authored by the highest contributor of a file.
- *MINOR* is the number of contributors who authored less than 5% of the code in that file.
- *OEXP* is the experience of the highest contributor of that file using the percent of lines he authored in the project at a given point in time.
- *EXP* is the geometric mean of the experiences of all the developers.

All of these metrics have been widely used in prior research literature [1, 5, 15, 18]. In addition, we use some "neighborhood" metrics inspired by BugCache [11]. BugCache uses co-commit history to identify files related to a buggy file which may also be buggy. This suggests the use of "co-commit neighbor" based process metrics. For these metrics we first find the list of files co-committed with a given file, weighted by the frequency of co-commit in a particular release; we then average the above metrics over this list.

[5]See [12] for details.

- We use the weighted average of a metric of the "commit neighbors" of a file: *NADEV*, *NDDEV*, and *NCOMM* are just the derived measures of *ADEV*, *DDVEV* and *COMM*.

All of these metrics are individually positively correlated with the existence of defects, and are thus reasonable candidates for inclusion in a prediction model; we rely on the aforementioned ridge estimator to handle any issues of multicollinearity. Software engineering data is highly skewed; following Menzies *et al.* [13], we log transform all variables to stabilize variance and improve prediction quality.

Bias-influence metrics Our goal here, as stated above, is to try to understand *the relative effects of size and different types of bias on prediction performance.* Bias arises because programmers link only *some* defect-fixing commits to defect reports, and fail to link others. Such bias can derive from the properties of the defects, the defect fixer or the files fixed.

Bird *et al.* [4] observed that developers are less likely to link severe defects than less-severe defects, and experienced defect-fixers will more often link their fix than inexperienced developers.

Experience and severity can thus be viewed as *bias-influencing properties.* We define *bias influence metrics* as measures of such bias-influencing properties. In this paper we consider 5 different bias-influencing properties, each with an associated bias influence metric (BI Metric), that have been discussed in prior literature [2, 4, 17].

- **Experience** Defect-fixer experience (measured as the percent of all commits to date that were made by the developer who fixed that bug).
- **Severity** Severity of the fixed defects (an ordinal measure).
- **Proximity** Proximity of the defect-fixing commit to the release deadline (days fromr defect resolution until the proximate future release).
- **Latency** The amount of time it took to fix a defect (days from reporting date to resolution date).
- **Cardinality** Size of commits (total of number of files committed in the defect-fixing commit).

Creating Biased Sub-datasets We use all of the above BI Metrics to create biased samples. For generality, we consider both directions of bias, for all sources of bias. Thus, instead of assuming that experienced developers would be more likely to link their fixing commits, or that the commits that fix severe defects are less likely to be linked, we study both directions: how is prediction performance affected when experienced developers are *more* likely to link their commits, as well as when they are *less* likely to link their commits; likewise we also consider the cases of severe defects being *more* likely to be linked, *and* less likely to be linked.

For each bias-influencing property, we partition our set of fixing commits using the median of the corresponding BI Metric. This gives us two sets: Lower containing the fixing commits with BI Metric $<$ median BI Metric, and Higher containing the fixing commits with BI Metric $>$ median BI Metric. We add the remaining fixing commits (with BI Metric exactly equal to the median BI Metric) to either Lower or Higher, whichever is smaller. This fairly coarse split is done to keep the experimental combinatorics under control; even with this simplification, we already have millions of sub-samples available for training.

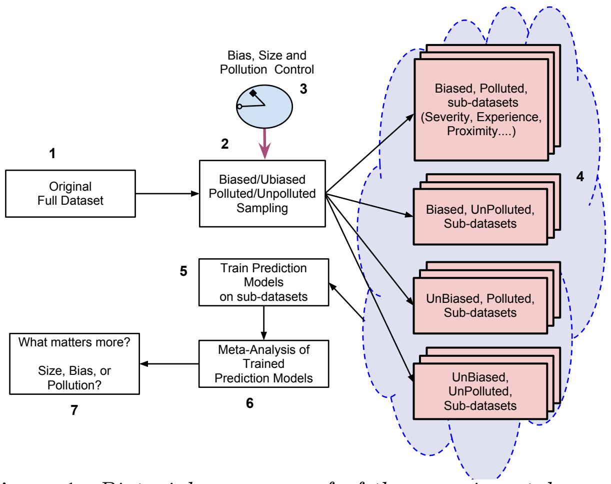
Based on this split, we choose our biased subsamples. Our approach is to select subsamples with varying levels of linking probability from LOWER and HIGHER. So for each of LOWER and HIGHER, we vary the probability of linking $p \in \{0.0, 0.1, 0.2, \ldots 1.0\}$. Thus, when $p = 0.2$, and we are considering LOWER, we assume that 20% of the fixing commits from LOWER are linked. These 11 discrete probabilities can be used to select 11 different samples for each of LOWER and HIGHER, designated as $\{\text{LOWER}_p\}$ and $\{\text{HIGHER}_p\}$, (for each value of p) randomly chosen from LOWER and HIGHER. During the sampling all the unselected fixing-commits are considered as missing links.

We then find the unions of all possible cartesian pairs of $\{\text{LOWER}_p\} \cup \{\text{HIGHER}_p\}$ (121 pairs) to study different ranges of bias. We discard the base set $\{\text{LOWER}_{0.0}\} \cup \{\text{HIGHER}_{0.0}\}$ as we need at least one defect-fixing commit to train a model. This approach allows us to study the bi-directional bias impact (when LOWER is less likely to link than HIGHER, as well as when LOWER is more likely) of each source of missing links. In order to compare the relative effects of size and bias, for each biased set of links $B_l = \{\text{LOWER}_p \cup \text{HIGHER}_p\}$ we also *uniformly randomly* sample an *unbiased* set of links U_l from the entire pool of fixing commits of the *same* size $|B_l|$.

Pollution Effects Besides possibly creating biased samples, missing links may *pollute* the data with false negatives: We therefore also examine the impact of POLLUTION, by sampling sub-datasets both with and without pollution. We create unpolluted sub-datasets by discarding any false negatives from the sampled sub-dataset, *i.e.*, any files known to be defective, but not selected based on the biased sampling procedure, are discarded from the training set. To study the impact of POLLUTION, we created polluted sub-datasets by labeling defect-fixing commits as non-defective when they are not chosen by the biased sampling procedure. We can then compare the impact of POLLUTION against the performance of an unpolluted dataset.

Finally, after sampling the defect-fixing commits with the four possible combinations (biased and polluted, biased and unpolluted, unbiased and polluted, unbiased and unpolluted) we derive the set of defective files based on our sampled defect-fixing commits. Any file that appears in one of the sampled fixing commits are considered as defective, otherwise defect-free. We can then build models on these newly labeled (coming from different combinations of bias and pollution) dataset. We replicate the entire process 100 times, to reduce the risk of random variation, and average the performance measures over all runs.

Summary of sampling procedure Figure 1 illustrates how we sample to create sub-datasets. The sampling process is adjustable; it can be tuned to select sub-datasets based on different settings described above; in addition, it can sample with or without pollution. Finally, the procedure can also sample without any bias, to create sub-datasets of varying sizes. Given all of the combinations of bias & pollution, all of the product-release pairs, and our 100-fold replication, this sampling procedure eventually creates about 17 million sub-datasets. These sub-datasets are used to train prediction models. We then evaluate the effect of bias, pollution, and size on prediction model performance.



Bias, Size and Pollution Control
3
1
Original Full Dataset
2
Biased/Ubiased Polluted/Unpolluted Sampling
Biased, Polluted, sub-datasets (Severity, Experience, Proximity....)
4
Biased, UnPolluted, Sub-datasets
5
Train Prediction Models on sub-datasets
UnBiased, Polluted, Sub-datasets
What matters more? Size, Bias, or Pollution?
Meta-Analysis of Trained Prediction Models
UnBiased, UnPolluted, Sub-datasets
7
6

Figure 1: *Pictorial summary of of the experimental procedure. We begin with a highly linked dataset (1) and selectively sample (2) under controls for bias, pollution and size (3) to create a cloud (4) of sub-datasets, with and without pollution, and bias, as well as an entirely unbiased sub-datasets. These sub-datasets are used to estimate prediction models (5); the results of these prediction models are then subjected to multiple regression meta-modeling (6) to tease apart the effects of pollution, size, and bias (7)*

Evaluation Our evaluation is a two step process. First, we a build a model for each training release, using the sub-dataset, and evaluate the model on the corresponding test release. It should be noted that the test release data is used in full, without sampling.

There are several different approaches to evaluating prediction models. We discuss them in general terms; details have been discussed in several prior publications. Precision/Recall and F-score are the traditional performance measures. Precision and recall have a natural trade-off, and one can choose a high-recall/low-precision combination, or the reverse. The precise value is based on a particular threshold of predicted probability of defects. These measures are also dependent on the proportion of defects; thus if most files are defective, even random choice will give relatively high precision.

There are also several threshold-independent, non-parametric methods. First, is the area under the ROC curve, AUC, which evaluates performance independent of threshold or defect occurrence rate. However, AUC doesn't directly consider the cost of methods such as inspection. Arisholm *et al.* [1] suggest that using threshold-dependent measure such as PRECISION, RECALL and F-MEASURE may not be suitable for software engineering data, since inspection cost depends on defect *density.* They recommend the use of AUCEC (area under the cost -effectiveness curve), and specifically AUCEC at 10% (AUCEC_{10}) and 20% (AUCEC_{20}) source lines of code (SLOC) to compare model performance. It should be noted however, AUCEC is just one model of cost; *e.g.* it does NOT consider the cost of false negatives. If false negatives matter, AUC in fact might be a better measure.

For our purposes, we consider threshold-dependent, threshold-independent, and AUCEC based measures. Specifically, we use F-MEASURE at 0.5 cutoff (F_{50}) to give the reader some idea about the model performance when evaluated in traditional cutoff based settings. Following Arisholm *et al.*, we stress that such a threshold-based measure may give an erroneous impression about the true performance of the model due to the class imbalance (only a small portion of our files are reported as defect prone) that is common in bug-fix datasets. So, we also measure performance using AUC, as well as AUCEC$_{10}$ and AUCEC$_{20}$. We replicate the entire process of random sampling and model building 100 times and then average over these runs to obtain the average model performance for each $\{\text{Lower}_p \times \text{Higher}_p\}$ and Pollution combination (Pollution turned off and on).

Meta-modeling Our goal is to use the models trained over the millions of samples of varying bias & size is to determine the relative effects of size and bias on performance. We do this by using a *meta*-model to model the performance of the resulting millions of *prediction* models; this meta-model essentially gauges the degree to which Bias, Pollution and Size influence the performance of the millions of prediction models. Thus, the *meta-response* of this meta-modeling step is the performance of the prediction of prediction models. F$_{50}$, AUC, AUCEC$_{10}$ and AUCEC$_{20}$ of the learned prediction models are all used as (meta) responses (measures of performance) in our meta-analysis, while Bias, Pollution and Size are the (meta) predictor variables. We build one meta-model for *each training-test release pair.* So, for each training release, we build an ensemble of samples of varying Bias, Pollution, Size, as described above. We then train models for each sample, testing on the next release to find AUC, F$_{50}$ etc. We use the prediction performance of these models as a response to our meta-model.

We validated our meta-models using standard OLS diagnostic techniques. All of our models exhibited reasonably high R^2. For AUC, AUCEC$_{10}$ and AUCEC$_{10}$ we observed a median R^2 of around 0.6, while for the F-MEASURE models, we observed an R^2 over 0.7. We checked for influential points and excessive heteroscedasticy through visual examination of the regression diagnostic plots. For influential points we looked for excessive separation or high Cook's Distance [6]. We also used a metric defined by Lindeman, Merenda, and Gold, (LMG) to measure the impact of the variables [9]. As described in detail in the next subsection, LMG provides a robust way to measure the relative importance of predictor variables on a response. LMG permutes the sequential sum of squares to yield an order insensitive degree of variance explained for each of the variables. The use of LMG provides resilience against the impact of multi-collinearity. This process is quite compute-intensive: building prediction models for the 17 million sub-datasets, over 85 releases, and the meta modeling phase, consume about 180 hours of time on 12-Xeon (3.0 GHz) core, 96 GB workstation.

We identify the percentage of variance explained by each of these variables using the R package Relaimpo [9]; The Relaimpo package computes the aforementioned "LMG" statistic which we use to measure the impact of the meta variables on the performance of the prediction models.

LMG measure of influence In an OLS multiple regression setting the Relaimpo package computes a statistic called LMG for each predictor. LMG is a measure of the influence that a predictor has on a response in a dataset. Using the LMG statistic, we can compare the effects of Size, Bias, and Pollution on the performance of prediction models. We now present our reasons for using LMG.

Ordinary Least Squares (OLS) linear regression models can be used to evaluate the relationship between a set of predictors $x_1, \ldots, x_p$ and a response y. We write the model form of the relationsip as

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p + \epsilon$$

where each β_i represents an estimated coefficient for the relationship between x_i and y. If β_i is positive and significant then we can infer that a β_i unit increase in x induces a one unit increase in y.

We were primarily interested in the magnitude of impact on the variance, rather than the direction of the influence. Bias may improve or retard a particular model performance measure; we would be interested in either, equally. Consequently, we are interested in the impact that each regressor has on the variance of the model response.

Typically one measures the variance in the response of an OLS model using the coefficient of determination, denoted by R^2. R^2 is interpreted as the percentage of variance explained in the response by the predictors, and is computed as the proportion of model sum of squares over the total sum of squares (SS)

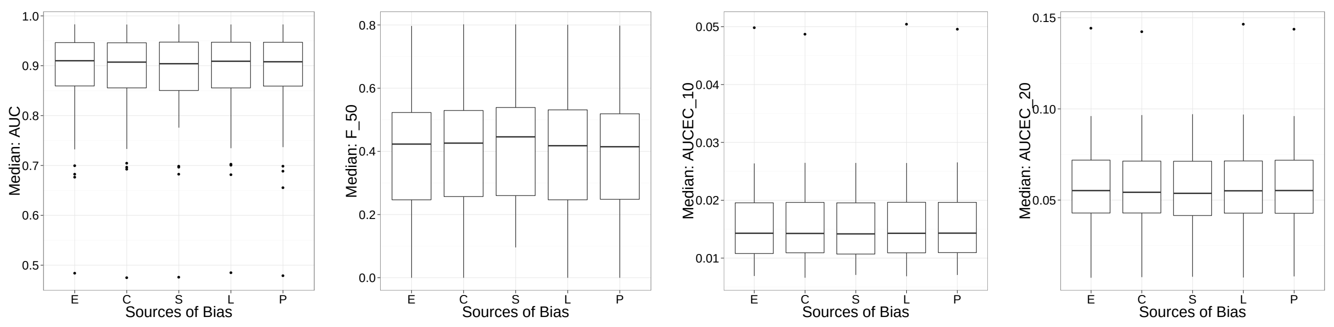
$$R^2 = \frac{\text{Model SS}}{\text{Total SS}} = \frac{\sum_{i=1}^{n}(\hat{y} - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where y_i is the given value of response in the i^{th} sample, $\bar{y}$ is the sample mean, and $\hat{y}$ is the predicted response from the OLS model, from the values of predictors in the i^{th} sample. Decomposing the sum of squares will allow us to compute the impact of each regressor on the variance of the response.

We can imagine a simple method for decomposing the sum of squares as follows: We add each predictor of interest in turn to the model and compute the additional variance explained by each new predictor. This will yield a particular decomposition of the sum of squares. A decomposition determined this way is referred to as a *sequential sums of squares.* No matter which order we choose to enter variables, every sequential sum of squares of the same set of predictors will yield the same *total* SS. Practically, the explained variance is typically computed by performing an ANOVA analysis.

Unfortunately, this procedure is sensitive to the order in which the variables are added to the model; Although we will always obtain the same total SS, the sequential decomposition may not be consistent across each variable. To see why, consider two predictor variables x_1 and x_2 that are not fully independent, *viz.*, each variable has both an indepenent component and a shared component. Then some part of x_1 can be explained by x_2, and the intersection will impact the response y simiarly for either variable. If we initially compute the sum of squares for the model containing only x_1, x_1 will account for all of the variance that is explained both by its independent component, and the component shared with x_2. When we add x_2 to the model, only its independent component adds additional variance explained as the variance owing to their collinearity is accounted for in x_1. Reversing the order will attribute the joint variance now to x_2 instead of x_1. This order-dependence makes this simple analysis unsuitable for decomposing the variance explained in our setting.

LMG is an order-independent method that calculates the decomposition of variance for all possible variable orderings and then computes the mean impact of each variable on the response. LMG is thus able to correct for order-dependency and provides a uniform measure of the impact of each predictor on the explained response. We calculate the raw LMG,



(a) Median AUC (b) Median F_{50} (c) Median AUCEC_{10} (d) Median AUCEC_{20}

Figure 2: *Median of Performance for different bias sources. E for* EXPERIENCE*; C for* CARDINALITY*; S for* SEVERITY*; L for* LATENCY*; P for* PROXIMITY

which calculates each variable's influence as a proportion of the total variance in the response.

Summary: To recap, by reference of Figure 1: the generated sub-data sets in Figure 1. (labeled **4**) (controlled for size, bias, pollution etc) are used to build prediction *models*. The performance of these prediction models is evaluated. We now have a collection of data, one for each prediction model, which capture the SIZE, BIAS, and POLLUTION of the data that went into that model, and the measured performance. Thus, for each model, we have one element of a meta-dataset. We now use ordinary least squares *meta-modeling* (step **6** in figure) on this dataset, and use LMG to tease apart the effects of SIZE, BIAS, and POLLUTION.

4. RESULTS

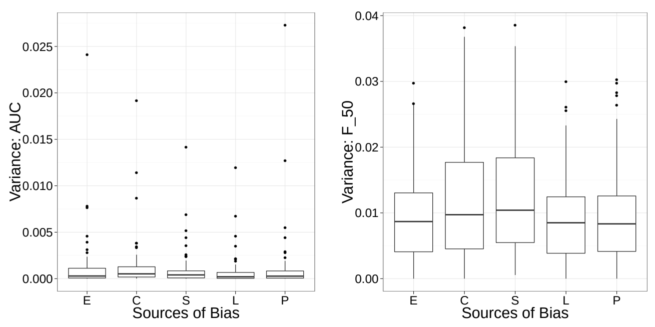
We begin by investigating the impact of different sources of bias on prediction performance.

RQ 1: Do different sources of bias have different impacts on prediction performance?

The raw data on the performance of the various models varies quite a bit due to different reasons: bias type, bias degree (H/L), release cycle, as well as sample size. We deal with the relative effects size and bias subsequently. For now, to focus on the effects of bias type, rather than bias degree or release cycle, we focus on summary statistics (median and variance *per release*, while bias degree varies).

For each release, and for each type of bias, as we allow the bias degree to vary, both the corresponding sample size and the degree of pollution will vary; so we expect to see some variation in the performance of models trained on each sample. So within each type of bias, we calculate the *median* and *variance* of performance *per release* as bias degree varies. These two statistics give us a sense of the range of performance that can be observed for each bias type, as its bias degree varies. The bias types in our case could be one of EXPERIENCE, CARDINALITY, SEVERITY, LATENCY and PROXIMITY. Since the goal here is to check whether different sources of bias have different effects (and thus if any of these sources of bias are a greater threat than any other) we consider only biased, *and* polluted sub-datasets, infected with bias of each type.

Figure 2 plots the median of prediction performance for different sources of bias. The figure shows hardly any difference in the distribution of median performance across different sources of bias. Our findings suggest that none of the observed variations are statistically significant. A non-parametric Kruskal-Wallis test also failed to reject the null hypothesis that the distributions are the same.



(a) AUC variance (b) F_{50} variance

Figure 3: *Variance of Performance for different bias sources. E for* EXPERIENCE*; C for* CARDINALITY*; S for* SEVERITY*; L for* LATENCY*; P for* PROXIMITY

In addition to analyzing the difference in median performance, we examined the stability of prediction performance using the variance of the performance measures *per release*. Figure 3a and figure 3b presents the variance of AUC and F_{50} for different sources of bias across releases. For brevity we do not present AUCEC_{10} and AUCEC_{20} as they are closely resemble the AUC plot (figure 3a). The figures suggest that the threshold-dependent performance measure F_{50} is more sensitive to different sources of bias than the threshold-invariant measures such as AUC and AUCEC. A Kruskal-Wallis test also failed to confirm any significant difference of variances between different sources of bias. We observe similar findings for AUCEC_{10} and AUCEC_{20}. It is clear that there is considerable variation in the median of the performance within each bias type. We attribute this variation in training sample sizes across releases; some releases have more defects, and thus more training data, than others. We return to the subject of sample sizes again below.

Bias Effect These two findings together have an important implication. In general, *across releases*, performance as measured by AUC, AUCEC_{10} and AUCEC_{20} varies (fig 2); these models, however, perform quite well, the median of AUC is about 0.9 and the median of F_{50} is around 0.4. The median of AUCEC_{10} is around 0.015, a figure that is 3 times 0.005, which corresponds to inspecting random lines[6], and the AUCEC$_{20}$ median figure is around 0.05, or about 2.5 times the random rate of 0.02. However, as we examine variation in performance, *within a release*, for models trained with sub-datasets with varying BIAS and POLLUTION, arising from different sources of bias influence, the variation in performance of prediction is *very limited* (Figure 3) for the non-parametric measures AUC, as well as for AUCEC$_{10}$ and AUCEC$_{20}$ (not shown)[7]. Furthermore, the effect on performance, and the variance in performance across different sources of bias, is also similar.

Different sources of BIAS have very similar effects on performance; furthermore, the effect of varying rates of BIAS is also minimal on non-parametric measures of performance, for all sources of BIAS.

This is an unexpected result; data that is selectively missing for different reasons might be expected to affect the performance differently, and different rates of bias might also be expected to affect performance. Nevertheless, we still can see that each source of missing links induce *some*, within-bias-source, variance in prediction performance. This variance can be attributed to varying SIZE and BIAS and the POLLUTION of the sample space with false negatives. We therefore study the impact of SIZE, BIAS and POLLUTION separately.

RQ 2: Considering BIAS, POLLUTION, and SIZE, which aspect of missing links affects prediction models the most?

When studying the relative effects of size and bias, we found a high degree of collinearity between size and bias in our meta-models. Multi-collinearity, and the resulting difficulties in interpreting models with potentially high variance inflation factors, casts doubt on the stability and validity of such models. Therefore, we chose to focus on a simplified, categorial measure of bias: *bias polarity.* Bias polarity admits two categories of possible, BIAS$_{\mathrm{L}}$ and BIAS$_{\mathrm{M}}$. BIAS$_{\mathrm{L}}$, negative bias polarity, represents all the samples where we have LOWER$_p$ > HIGHER$_p$, where LOWER$_p$ and HIGHER$_p$ represent the probability of linking from LOWER and HIGHER respectively. In case of EXPERIENCE bias, this would indicate that inexperienced developers are more likely to link than experienced developers. Likewise BIAS$_{\mathrm{M}}$, positive polarity represents all the samples where LOWER$_p$ < HIGHER$_p$. The samples are coded so that BIAS$_{\mathrm{L}}$ (resp., BIAS$_{\mathrm{M}}$) take the value 1 if the sample has negative (resp., positive) bias polarity. They both take the value 0 if the sample has neutral bias, *viz.*, LOWER$_p$ = HIGHER$_p$.

Arguably, for the BI METRICS we consider, polarity is a reasonable categorical abstraction. Concern about bias can be stated in categorical terms such as *Are more/less experienced developers more/less/equally likely to link bugs?*, or *Does linking rate for fixes that include more changed files lower? or higher? or the same?*, or *are more/less severe bugs less/more/equally likely to be linked than less severe bugs.* Our abstraction sheds light on whether such categorical biases affect prediction performance.

Our ensemble of samples includes both biased and unbiased samples. Our set of unbiased samples will include both uniformly randomly picked unbiased samples, as well as samples where LOWER$_p$ = HIGHER$_p$. We also add POLLUTION as a treatment, with two levels, polluted and unpolluted, as described above. Those are the predictors; the (continuous) response variables are AUC, AUCEC$_{10}$, AUCEC$_{20}$ and F$_{50}$ as response.

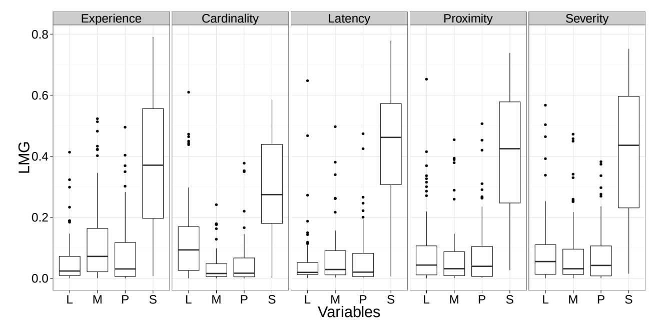
To find the impact of different variables we use LMG as described in Section 3. Figure 4 shows the LMG impact of BIAS, SIZE and POLLUTION variables on different measures of prediction performance. BIAS$_{\mathrm{L}}$ and BIAS$_{\mathrm{M}}$ are marked as L and M respectively on the x-axis. P and S represent POLLUTION and SIZE respectively. Since there is one prediction model for each of the training releases, we get a range of LMG values bias, size, and pollution; these are shown as box plots labeled below with L, M, P and S as just explained. The figure depicts a surprising trend: *for* AUC*,* SIZE *matters much more than both* BIAS$_{\mathrm{L}}$ *and* BIAS$_{\mathrm{M}}$. We ran two sample *paired* Wilcox test (with alternative hypothesis set to "SIZE has a larger LMG impact than either BIAS$_{\mathrm{L}}$ or BIAS$_{\mathrm{M}}$ ") on each pair , and corrected the p-values for multiple hypothesis test using Benjamini-Hochberg correction. For all sources of missing links, we observed that the impact of SIZE is statistically significantly more than the impact of either BIAS$_{\mathrm{L}}$ or BIAS$_{\mathrm{M}}$ with very low p-values $p < 0.001$.

When considering AUC, SIZE is much more important than BIAS polarity. Focusing effort on collecting more samples may mitigate much of the impact of BIAS polarity.

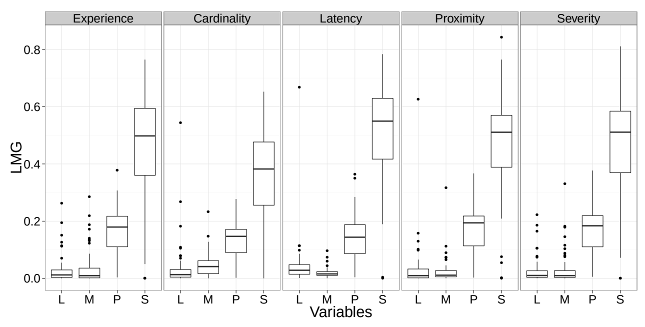
For AUCEC$_{10}$ and AUCEC$_{20}$, however, the impact of SIZE doesn't dominate that of BIAS. A one-sided Wilcox test with alternative hypothesis "SIZE has *less* impact than either BIAS$_{\mathrm{L}}$ or BIAS$_{\mathrm{M}}$ ", failed to establish any statistical significance of BIAS against SIZE after Benjamini Hochberg correction; all of the p-values were greater than 0.40. In fact, testing the alternative hypothesis "SIZE has *larger* impact than either BIAS$_{\mathrm{L}}$ or BIAS$_{\mathrm{M}}$ ", we observed that SIZE has a larger impact than both BIAS$_{\mathrm{L}}$ and BIAS$_{\mathrm{M}}$ for LATENCY ($p < 0.001$). This result holds for both AUCEC$_{10}$ and AUCEC$_{20}$. SIZE also dominates both BIAS$_{\mathrm{L}}$ ($p = 0.004$) and BIAS$_{\mathrm{M}}$ ($p = 0.001$) for AUCEC$_{20}$ when considering PROXIMITY, and BIAS$_{\mathrm{M}}$ for AUCEC$_{20}$ ($p = 0.014$) when considering CARDINALITY, and BIAS$_{\mathrm{M}}$ for both AUCEC$_{10}$ ($p = 0.048$) and AUCEC$_{20}$ ($p = 0.001$) when considering SEVERITY. This suggests that SIZE is as important as BIAS even when evaluating AUCEC.

With respect to F$_{50}$, we again observed a dominance of SIZE over BIAS. In all of our comparisons we found a statistically significantly higher impact ($p < 0.001$) of SIZE, compared to either BIAS$_{\mathrm{L}}$ or BIAS$_{\mathrm{M}}$. In fact, it is clear from the figure that, for F$_{50}$, SIZE, moreso than AUC, dominates both BIAS$_{\mathrm{L}}$ and BIAS$_{\mathrm{M}}$.

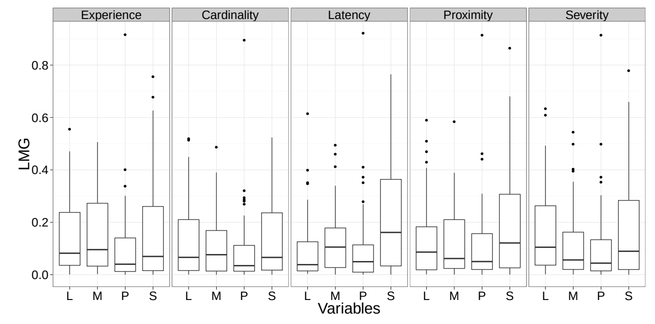
[6]If we inspect lines randomly, over many samples, we expect to discover bugs at the same rate we inspect, *viz.* 10% of the defects for inspecting 10% of the lines; thus the area under the resulting diagonal line would be 0.1 times 0.1 times 0.5
[7]The parametric measure F$_{50}$ does show a much more substantial effect, but this is due to 50% threshold parameter not always being ideal; we discuss this later



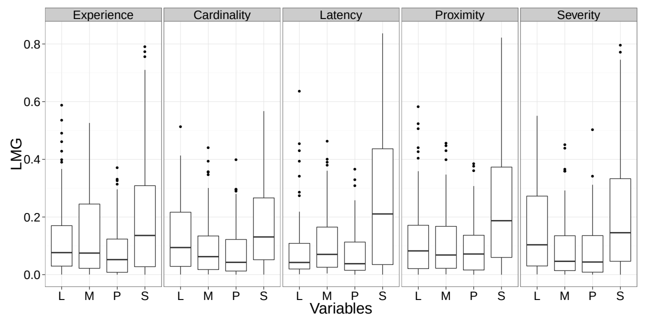
(a) Impact on AUC



(b) Impact on F_{50}



(c) Impact on AUCEC_{10}



(d) Impact on AUCEC_{20}

Figure 4: *Impact of different aspects of missing links. L stands for bias polarity to low values (*BIAS_L*); M bias polarity to higher values (*BIAS_M*); P stands for pollution; all of those are categorical variables. S stands for size, the sole numerical variable*

For the IR performance measures AUC and F_{50}, the effect of SIZE strongly dominates that of BIAS. For the AUCEC performance measure, SIZE has as much of an influence as BIAS polarity.

We have discussed the impact of SIZE and BIAS, however, missing links would inevitably introduce POLLUTION. As stated above, we introduce POLLUTION as an additional treatment in our meta-model. Figure 4 shows, in addition to BIAS and SIZE, the impact of POLLUTION on performance. The figure shows that POLLUTION matters much more for the threshold-dependent measure F_{50}, than the threshold-invariant measures AUC and AUCEC. We compared the impact of POLLUTION with SIZE, BIAS_L and BIAS_M using a *paired* Wilcox test and corrected the p-values using Benjamini-Hochberg correction. Our findings suggest that for AUC, AUCEC_{10} and AUCEC_{20}, the impact of POLLUTION is typically neither significantly greater nor smaller than the impact of BIAS_L or BIAS_M. In fact POLLUTION showed a significantly smaller impact ($p = 0.016$) for *only* one case: against BIAS_L for AUC and CARDINALITY. In all cases, for AUC and AUCEC, POLLUTION showed statistically significantly less impact than SIZE (for AUC, $p < 0.001$, and for AUCEC the largest observed $p < 0.027$).

However, a similar comparison for the threshold-dependent measure F_{50} tells a different story. In this case, POLLUTION was dominant over both BIAS_L and BIAS_M, and was only outperformed by SIZE. A *paired* Wilcox test, using a one-sided alternative that "POLLUTION has greater impact than either BIAS_L or BIAS_M ", shows a statistical difference between the variables with a p-values ($p < 0.001$). A *paired* Wilcox test for the alternative hypothesis that "POLLUTION has smaller impact than SIZE ", however, confirms the dominant role of SIZE ($p < 0.001$).

For F_{50} performance, POLLUTION plays a more damaging role than BIAS polarity. SIZE still has the most impact.

5. DISCUSSION

While previous studies have considered bias and its implications, ours is the first detailed comparison of the effects of SIZE, BIAS and POLLUTION. We now discuss the implications.

Positive & Negative Bias Note in Figure 4 that the polarity of BIAS indicates varying impact on the prediction performance. In case of EXPERIENCE, (top left of the figure) with AUC as performance measure, linking by more experienced developers (the box plot labeled BIAS_M) has a stronger effect on performance. (a statistical test confirms this). But in the case of CARDINALITY, we see the opposite effect on AUC.

We also examined the effect of combining positive and negative bias. In general, we found that combining bias into a "directionless biased" treatment tended to lower the effect of bias ($p < 0.001$ for all pairwise comparisons of directionless BIAS with directional BIAS). In practice, however, it is not always easy to determine whether a given dataset has positive or negative bias. Our study suggests that direction does matter, and some effort, perhaps based on sampling, might provide useful information.

From Figure 4 we also notice that each BIAS may have an impact on the prediction performance. E.g., in case of EXPERIENCE and AUC as performance measure, linking by more experienced developers may be more important. A two-sample *paired* Wilcox test shows that the impact of BIAS_L on the AUC is significantly different than the impact of BIAS_M for CARDINALITY ($p < 0.001$) and EXPERIENCE ($p = 0.013$). For AUCEC, the direction of BIAS mattered only for SEVERITY ($p = 0.034$ for AUCEC_{10} and $p = 0.025$ for AUCEC_{20}). For F_{50}, direction mattered only for CARDINALITY ($p = 0.016$) and LATENCY ($p < 0.001$).

Pollution effect on $\mathbf{F}_{50}$ From Figure 3, it is clear that the F_{50} performance measure has a much greater variance than AUC. F_{50} also has a lower variance than the AUCEC measures. It is also clear from Figure 4 that F_{50} is quite strongly affected by pollution, when compared to other prediction performance measurements.

F_{50} assumes a 50% threshold on the predicted probability from the underlying logistic regression prediction models. We hypothesized that performance based on this fixed threshold is highly sensitive to the precise training set. We expect that different sizes and biases in the training set will result in variable performance. To investigate this further, we measured the F_{50} score on the full datasets for each release. A two-tailed, two-sample Kolmogorov-Smirnov test showed that F_{50} over the unsampled data and F_{50} over the sampled data are indeed different, *i.e.*, sampling affects the F_{50} performance. To reduce the sample-dependent effect on F_{50}, we tried an alternative approach: for each training data set, we chose the threshold that *maximized* F-MEASURE performance on the training set. This is a feasible approach, since no test set information is used. We call this F-MEASURE value F_{tr_max}. We calculated F_{tr_max} for both sampled sub-datasets and the full dataset. A statistical test failed to reject the null hypothesis that the distribution of the F_{tr_max} scores in the sub-datasets was different from the F_{tr_max} scores in the full dataset. This provides a plausible explanation for the instability in F_{50}.

Implications Overall, our meta-analysis shows that SIZE influences prediction performance at least as much as BIAS polarity and POLLUTION. When considering inspection-oriented applications, all factors matter. The AUCEC results suggest that one should not only try to maximize the size of the training data, but also to the extent possible, strive to obtain unbiased and unpolluted samples. AUCEC, however is not the last word. For example, it doesn't account very well for the cost of false negatives. When these matter, AUC might be a more suitable measure. In this case, our data suggests that SIZE has a *much stronger influence* on performance; therefore efforts to obtain larger training sets might be a more effective way to improve performance. Since, in general, it is difficult to determine the nature and extent of BIAS polarity, this finding is a reassuring and positive result: *if better* AUC *is a goal, and if there is concern about the polarity of bias, a larger training sample is likely to help.*

Finally, we note that our study focused on BIAS **polarity**, rather than the degree of bias. Our results indicate very strongly that bias *polarity* matters far less than size; if large sample sizes are available, our results suggest bias polarity *per se* is not a major concern that it is very likely that trained prediction models would perform reasonably well, at least from from an AUC or F_{50} perspective. Indeed, the effect of bias is *so strongly* confounded by the effect of size that we found it difficult to tease apart the effect by linear modeling, and this remains for future work.

6. THREATS TO VALIDITY

Choice of Metrics Our prediction models use popular process metrics. We did not use any code metrics; generally speaking, process metrics outperform code metrics [1, 16, 19, 20]. Our models performed well: the median AUC was around 0.9, and median F_{50} was about 0.5 for the full dataset, on par to what is reported in the existing research [1, 22].

Using Logistic regression We use LR to predict the defect-proneness of files. LR is very widely used [1, 15, 18, 22]; Moreover, prior research indicates that the choice of proper metrics matters more than the learning techniques used, and that LR has good performance [1].

Data Quality Instead of relying on automated extraction of links from commit logs, we rely on a high fidelity linking process available in JIRA. All of our projects have very high linking rates, with a median linking rate about 80%. This is well above the typical linking rate of under 50% reported in prior research [4] using traditional heuristic based linking technique [7, 8]. The Jira data has link bias for SEVERITY (80% median link rate for more severe vs. 75% for less severe) and LATENCY (80% vs 78%); it is unbiased for PROXIMITY. Nevertheless, the high linking rate allows us to subsample and introduce both positive and negative bias, and pollution, to study the impact on prediction performance.

7. CONCLUSION

Several publications in the last 3-4 years have highlighted the presence of significant bias in bug-fix datasets. This has led to widespread concerns, reported in several papers, that biased datasets would lead to under-performing, even misleading, prediction models of limited practical value. We investigated this issue in depth using simulated sampling with bias from high-quality dataset, and found clear evidence that a) the *type* of bias has limited impact i on prediction results, and b) the effect of bias is strongly confounded by size, and c) Bias *polarity* has a relatively small effect effect when compared to size for the AUC and F_{50} measures, and is comparable to size for the AUCEC measures. Indeed, we found that the effect of bias is difficult to tease apart from size, and this remains a challenge for future work. Thus the bias-effect results reported earlier [4] may indeed be due to diminished sample size in the biased samples. Our work does strongly suggest, however, that *even if* there are concerns about bias polarity in a potential training sample of defect repairs, such a sample could still be used to train a prediction model, as long as it is large; the resulting prediction performance is likely to be boosted more by the size of the sample than it is hindered by any bias polarity that may exist.

8. ACKNOWLEDGMENTS

This research was conducted while Israel Herraiz was visiting UC Davis, thanks to the funding provided by the Ministry of Education of Spain ("José Castillejo" grant no. JC2011-0093). Prem Devanbu, Foyzur Rahman and Daryl Posnett were supported by NSF, Award Number: 0964703. Daryl Posnett was also supported by the Air Force Office of Scientific Research, award FA955-11-1-0246.

9. REFERENCES

[1] E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *JSS*, 83(1):2–17, 2010.

[2] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein. The Missing Links : Bugs and Bug-fix Commits Categories and Subject Descriptors. In *Proceedings of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2010)*, volume 2 of *FSE '10*, pages 97–106. ACM, 2010.

[3] C. Bird, A. Bachmann, E. Aune, and J. Duffy. Fair and balanced?: bias in bug-fix datasets. In *Proceedings of the the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on The Foundations of Software Engineering*, 2009.

[4] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu. Fair and balanced?: bias in bug-fix datasets. In *Proceedings of the the 7th FSE*, pages 121–130. ACM, 2009.

[5] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. T. Devanbu. Don't touch my code!: examining the effects of ownership on software quality. In T. Gyimóthy and A. Zeller, editors, *SIGSOFT FSE*, pages 4–14. ACM, 2011.

[6] J. Cohen. *Applied multiple regression/correlation analysis for the behavioral sciences.* Lawrence Erlbaum, 2003.

[7] D. Cubranić and G. C. Murph. Hipikat: recommending pertinent software development artifacts. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 408–418, Portland, Oregon, 2003. IEEE Computer Society Press.

[8] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings of the International Conference on Software Maintenance*, pages 23–32, Los Alamitos CA, September 2003. IEEE Press.

[9] U. Grömping. Relative importance for linear regression in r: the package relaimpo. *Journal of Statistical Software*, 17(1):1–27, 2006.

[10] S. Kim, H. Zhang, R. Wu, and L. Gong. Dealing with noise in defect prediction. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 481–490. ACM, 2011.

[11] S. Kim, T. Zimmermann, E. Whitehead Jr, and A. Zeller. Predicting faults from cached history. In *Proceedings of the 29th ICSE*, pages 489–498. IEEE Computer Society, 2007.

[12] S. Le Cessie and J. Van Houwelingen. Ridge estimators in logistic regression. *Applied statistics*, pages 191–201, 1992.

[13] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE TSE*, 33(1):2–13, 2007.

[14] A. Mockus and L. G. Votta. Identifying reasons for software changes using historic databases. In *ICSM '00*, page 120, Washington, DC, USA, 2000. IEEE Computer Society.

[15] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.

[16] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In W. Schäfer, M. B. Dwyer, and V. Gruhn, editors, *ICSE*, pages 181–190. ACM, 2008.

[17] T. H. Nguyen, B. Adams, and A. E. Hassan. A case study of bias in bug-fix datasets. In *Proceedings of WCRE*, pages 259–268, 2010.

[18] D. Posnett, V. Filkov, and P. Devanbu. Ecological inference in empirical software engineering. In *ASE'2011*, pages 362–371. IEEE, 2011.

[19] F. Rahman and P. Devanbu. How, and why, process metrics are better. `http://www.cs.ucdavis.edu/research/tech-reports/2011/CSE-2012-33.pdf`, 2012.

[20] F. Rahman, D. Posnett, and P. Devanbu. Recalling the "imprecision" of cross-project defect prediction. In *the 20th ACM SIGSOFT FSE*, pages –. ACM, 2012.

[21] R. Wu, H. Zhang, S. Kim, and S. C. Cheung. ReLink : Recovering Links between Bugs and Changes. In *Proceedings of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2011)*, 2011.

[22] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, PROMISE '07, pages 9–, Washington, DC, USA, 2007. IEEE Computer Society.