What is "Enough" Quality for Data Repositories?

Tim Menzies, Adam Brady
Lane Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, West Virginia, USA,
tim@menzies.us, adam.brady@gmail.com

***Abstract*—BACKGROUND: Some project data is only approximately correct. This may limit our ability to draw conclusions from project data.**

AIM: To be able to reason about software projects, despite inaccuracies in the data.

METHOD: For ten data sets, numerical data from the project was discretized into fewer and fewer bins. A case-based planning algorithm is applied to all versions of the data sets. The planner seeks changes to a project that moves it from regions of (e.g.) large effort estimates to other regions with lower effort estimates. The effects of descretization was

***Index Terms*—component; formatting; style; styling;**

I. Introduction

Data *could* be collected, accurate to seven decimal places, by N independent consultants, and then extensively auditted prior to addition to a repository. Such a collection procedure may be prohibitively expensive. Is there some way to define a data quality program that balances data collection *cost* against the $benefits$ of having data at different levels of quality?

Recently, Brady [**?**] has proposed an algorithm for determining if data in a repository is of sufficient quality to support business decision making. The algorithm, called Δ-resiliancy, works as follows:

- Given some sample data $data_0$, keep throwing away details to generate $data_1$, $data_2$ etc until something breaks; e.g. our ability to make business decisions start to degrade.
- The data set $data_{i-1}$ generated just *before* the breakdown at point i has *enough* quality.
- Design future data collection such that it collects data of quality $data_{i-1}$.

More formally, we define Δ-reiliancy as follows:

> *Definition 1: A data set has "enough" quality if the data noise is Δ and the inferences from that data set are not effected by noise up to the level of Δ.*

Note that Δ-resiliancy requires some technique for analyzing data since the point i at which this technique starts to fail is used to identify the point of $enough$ quality (at $i-1$). That is, without knowledge of *how* the data is being used, the above definition is not operational. Hence, Δ-resilancy is defined for repositories that have reached level five maturity (since, if a repository is below level five, then there are no active optimizations being generated from the data).

The algorithm supports a cost/benefit analysis of data collection; specifically, we need only collect data of $enough$ quality since more elaborate and expensive collection methods is superflous. The clear advantage of this approach is that it expresses quality in terms business users can understand; e.g. "if you grant these funds, then we will be able to detect out-of-control projects sooner". This is useful since, as discussed below, other algorithms for data quality are more algorithmic and do not express their conclusions at the business-level (hence, it may be difficult to use those algorithms to lobby for extra funding for data collection).

This paper conducts an in-depth study of Δ-resiliancy on six real-world data sets. Since Δ-resiliancy requires some operational definition of how data is used to make decisions, we will assume an instance-based reasoning decision framework. Instance-based reasoning is a general method for reasoning about (say) software engineering data [**?**], [**?**], [**?**], [**?**], [**?**] and is known to work for the kinds of complex data sets collected from real-world projects [**?**]. For this work, we use Brady and Menzies' W instance-based reasoning algorithm [**?**], [**?**], [**?**]. The results will be mixed:

1) The good news is that it is possible to reduce the level of detail within data without comprosmining the analysis technique. This, in turn, raises the possibility of extensive cost-savings in repositories (by avoiding needlessly over-elaborate data collection policies);
2) The other news is that the point at which data breaks down is not *data set* dependent by *instance* depedent. That is, we cannot declare that for one data set that "this is enough quality". Rather, we have to say that "for instances of a certain type within a data set, then this is enough quality.

The first result should encourage more research for algorithms like Δ-resiliancy. The second result tells us that data quality is not an issue that can be retired with a single paper. Rather, it is very complex issue that will require extensive further research.

The rest of this paper is structures as follows. XXX.

II. Background

Recent research results highlight the value of repositories of example problems. Such repositories are useful for (a) storing the results of case-study based research; (b) for documenting existing baseline results; (c) for one researcher to defend a claim that their new analysis method exceeds those baselines; (d) for other researchers to audit such claims.

Just as the machine learning community focused on the UC Irvine machine learning repository[1], entire research communities in software engineering have now formed around:

[1]UCI: http://archive.ics.uci.edu/ml

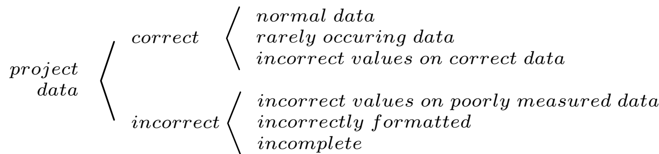


Fig. 1. Data quality definitions (gray cells denote different kinds of outliers). Adapted from Yoon & Bae [?].

- Numerous open-source repositories such as Bugzilla[2];
- The Software Infrastructure Repository (SIR)[3];
- The PROMISE repository of reproducible SE results[4];
- The ISBSG estimating benchmark suite[5].

Numerous publications are now based on this data. While the number of papers using ISBSG and Bugzilla data are not known, since the PROMISE and SIR repositories came on line in 2006 and 2001, their data has been used in at least 90 and 200 publications, respectively[6].

Sen et al. document a parallel interest in commercial *data warehouses*. They report that "data warehousing (DW) has experienced tremendous growth in the last decade ... it was cited as the highest-priority post-millennium project of more than half of IT executives". [?]. They define a *data warehouse process maturity model* in which, at level five, ensures that organizations can use their data to *optimize* their performance.

Given this increasining use of data repositories by researchers and industrial practitioners, it is important to certify the quality of that data. Software engineering data can contain large amouts of *noise* (signals not connected to the known target variables). Sen et al. note that managing poor data quality is a very expensive task requring "subject matter experts, who are knowledgeable about business as well as data, are often employed to define data cleansing rules and data quality metrics".

Figure 1 shows Yoon & Bae's hierarchy of data quality issues for software engineering [?] The SE literature explores this hierarchial with different levels of rigor. For example, much of the SE data mining literature discusses the problem of *missing data* (which Yoon & Bae would call "incorrect values"):

- Aranda & Venolia [?] audited bug reports at Microsoft by interviewing developers related to the reports. They found much missing information in the bug reports.
- This result was confirmed by Bird et al. [?] who, in a found many missing links (up to 50%) between change logs and bug reports were not linked.
- Yet another study confirmed the Aranda & Venolia & Bird result. Kim et al. [?] found 32% and 22% unlinked bugs in open source systems (Eclipse 3.1 and 3.4).
- Note that change that is not linked to a bug would result in an incorrect value on the classification of that change. Bird et al. caution that such incorrect values can degrade the performance of some predictor built from noisey data [?].

[2]BUGZILLA: http://www.bugzilla.org/installation-list/
[3]SIR: http://sir.unl.edu
[4]PROMISE: http://promisedata.org/data
[5]ISBSG: http://goo.gl/2AJCH
[6]SIR publications: http://sir.unl.edu/portal/usage.html

China			
	25%	Median	75%
DevType	0	0	0
PDR-AFP	0.1	0.1	0.2
NPDU-UFP	0.1	0.1	0.2
PDR-AFP	0.1	0.1	0.2
PDR-UFP	0.1	0.1	0.2
Enquiry	1	1	2.5
Input	1	1	4
Output	1	1	4
Resource	0.5	1	1
Duration	0.5	1	1
Interface	1	2	5
NAFP	1	2	6
Added	1	2	7
Changed	1	2	4
Deletec	1	2	7
File	1	2	5.5
Kemerer			
	25%	Median	75%
Duration	1	2	3
Hardware	1	1	1
Language	0.5	1	1
KSLOC	2	12.3	35
RAWFP	21	124	170
AdjFP	34.7	130	182
Finnish			
	25%	Median	75%
lnsize	0.03	0.05	0.115
prod	0.183	0.428	0.874
at	1	1	1
co	1	1	1
hw	0.5	1	1
FP	14	28	64

Fig. 2. Quartiles demonstrating the ranked difference between each unique feature value. Within each datasets, rows sorted on the median values. Data from http://promisedata.org/data.

Another issue related to "incorrectly formatted" or "incorrect values" from Figure 1 is that of *granularity*. It can readily be shown that data is collected according to a wide variety of standards. For example, consider the issue of *granularity* in a data set. How much detail is sufficient for including data in a warehouse? If data arrived rounded to the nearest integer, should we reject it and ask for more details?

While there may not exist domain-general answers to this question, it is easy to show that data is collected with very wide ranges of granularity:

- Take data from some public source (e.g. the PROMISE repository) and sort the columns of numeric data to form the list $C_{i,1}, C_{i,2}, C_{i,3}, ...$ for each attribute i.
- Next, find the differences between adjacent members in that list; i.e. $diff(i)_j = C_{i,j+1} - C_{i,j}$.
- Finally, sort those $diff(i)$ values and report their 25th, 50th and 75-th percentile.

The results on one such analysis from PROMISE data are shown in Figure 2. Several aspects of these results are worthy of comment. Firstly, there is enormous difference in data collection precision for different data values in that figure. Some attributes show *wide ranges* (i.e. have have very large median differences); for example:

- The $RAWFP$ attribute of $Kemerer$ has a median difference of 124, which is nearly as large as the entire inter-quartile for that attribute (170-21 = 149).
- The $AdjFP$ attribte of $Kemerer$ has a median difference of 130 which is also nearly as large as that attribute's inter-quartile range (182-34.7 = 147.3).

Are such large median differences are too large or too small? This paper argues that such questions are unanswerable *unless* we know the intended use of that data. That is, in this research, we say that data is of *enough* quality if it supports the inference to which it is intended.

Such a concept of *intent* lets us resolve other issues. Consider the *"flat"* attributes of Figure 2 (those where the 25-th percentile is the same as the 75-th percentile):

- In $China$, the $DevType$ attribute is *"flat"*;
- In $Kermer$, the $Hardware$ attribute is *"flat"*;
- In $Finish$, the at and co attributes are *"flat"*;

(Note that the variance of such *"flat"* attributes can be very small. Hence, many machine learning algorithms would simply ignore them since they do not offer significant differences between classes.)

Are such *"flat"* attributes of low quality? Is it worthwhile to inject more domain knowledge into such attributes, discover more distinctions, and make them unflat? Perhaps not- especially if the inferences made from this data do not require such *"flat"* data. As before, we are arguing that if the data supports the planned inferences, then the data is of sufficient quality.

Finally, observe how some of the data in Figure 2 is collected to a very exacting degree of procession:

- In the $China$ data set, the median difference between numeric column entries for some of the speciailized function points is very small; e.g. see the median difference of 0.1 for *PDR-AFP*.
- In the $Finnish$ data set, the median difference between numeric column entries is very small for the $insize$ measure (median value of 0.05).

Should we always encourage such high degrees of precision? Perhaps not since infinite precision would be infinitely expensive. Such exacting standards of precision are nonsensical for noisy data(since all they do is precisely measure some random value. This *can* be a major problem since many data sets in software engineering are very noisy [**?**]. However, repeating our theme, we would say that the numeric precision of the measurements are *enough* if they support the required inferences from the data. We would also say that the numeric precision of the measurements are *over-elaborted* if, after some rounding, the inferences do not change (this second statement is the basis of the Δ-resiliancy definition from the introduction.

III. Algorithms for Data Quality

A. Standard Methods

A standard method for data cleansing noise is to *prune* data items associated with noise [**?**], [**?**]. Recent research in *defect prediction* focuses on two different kinds of *row pruning*:

- *Outlier removal that deletes a minoirty of rows*: Kim et al. [**?**] use case-based reasoning to prune neighboring rows containing too many contradictory conclusions. Also Yoon & Bae [**?**] use association rule learning methods to find frequent item sets. In this framework, outliers are those rows with few frequent items. Their methods prune 20% to 30% of the rows.
- *Prototype selection that deletes most rows*: Turhan et al. [**?**] pruned away all but 10 training examples per test instance using nearest neighbor methods. In doing so, they dramatically reduced the false alarm rate of defect predictors being trained on data from other companies.

Recent reseach in *effort estimation* demonstrates the value pruning *both* row and columns:

- Chen et al. [**?**] found that removing up to 80% of the columns significantly improved PRED(30)[7] values from (approx) 20 to 60 %. The effect was particularly marked in smaller data sets (those with less than 30 rows).
- Kocaganeuli et al. [**?**] showed that when building an effort estimator for local data, then importing effort data from other organizations severely damages predictor performance. However, this damaging effect is removed if, prior to learner a predictor, they they pruned away neighboring rows with high variance in their effort values.

(As an aside, we note that while the techniques of Kocaganeuli et al. were evolved seperately to Kim et al, they are clearly analagous techniques. For another example of analagous research, see our work with Guo et al. where we explored a similar association-rule approach to detect anomalies in flight guidance systems [**?**].)

While row and column pruning can sometimes patch poor data quality, they are only heuristic methods that may not work on particular data sets. For example, for many years, we have tried column and row pruning to reduce the noise in this manner. The results have been disappointing. Jalali et al. report that row pruning reduce variance in our predictions [**?**]. As to column pruning, while it reduces the median performance variance somewhat (in our experiments, from 150% to 53% [**?**]), the residual error rates are still unacceptably large.

In summary, standard methods such as row and column pruning are not sufficienent to repair low-quality data.

B. Δ-Resiliancy

Row and column pruning are data cleansing methods for patching low quality data *after* it arrives in a repository. In terms of setting corporate policy on data quality, the preferred alternative is a *data collection quality standard*. Such a standard can be used by data providers to certify that their data has "enough" quality *before* it is added to a repository.

Recall from the above that if a repository reaches level five maturity, then it is used for some business optimziation task. Δ-resilinacy certifies that, up until some noise threshold Δ, that the functioning of that optimizer is *not* effected by uncertainty in values. Once that value is determined, then

[7] The percentage of estimated within 25% of actuals.

the task of data provideers is to demonstrate that the noise resulting from their data collection methods is *less than* Δ.

Consider a model that accepts N inputs $N_1...N_n$. Let the known values for N_i be the range $V_1,...V_v$:

- Since we seek a measure of noise across all N_i, we study the $rank$ of the values V_i using the normal ranking rules[8]
- Noise can change the value of that input N_i by some rank r. Given v values for N_i, then we can express that rank as the δ_i percentange $\frac{100*r}{v}$.
- Let Δ be the maximum δ_i for all optimizer inputs.

Then:

> *Definition 2: The outputs of a Δ-resiliant optimizer are not degraded when the majority of ranks in the optimizer inputs are changed by up to Δ%.*

One way to find Δ is via $descritization$ studies. *Equal frequency discretization* maps numerics N_i into B bins (labelled $1, 2, ..., B$), each of which contain equal number of values [**?**]. If an optimizer's inner processing requires precise numeric values, then we replace all values V_i in one bin with the median of all the values in that bin.

To compute Δ-resiliancy, we divide the model inputs into B bins, then execute the model. Next, we seek the smallest value B' such that:

1) Optimization results from inputs divided into $B'-1$ bins are no *worse* than B';
2) Optimization results from inputs divided into $B'+1$ bins are no *better* than B'.

B' offers a lower-bound on the level of detail required for the optimizer. It is superflous to collect data at any level of detail greater than B' since that extra level of detail does not improve the optimizers.

Also, B' offers an upper bound of the resiliance of the optimizer to noise. If we divide ranked values $V_1, V_2, ..V_v$ into B equal frequency bins, then each bin will have cardinality $\frac{v}{B}$. For the majoirty of entries in any bin, any uncertainity in rank (due to noise) less than $V/B/4$ will not change the bin of those values (see Figure 3). Hence, once B' is known, we say that the optimizer is resilant to rank changes dues to noise up to $\Delta = 100/B'/4\%$.

IV. Buidling Optimzers From SE Data

This definition of Δ-resiliancy is based around an optimizer that studies data in a repository to find ways to improve a business. Hence, as a pre-condition for operationalizing Δ-resiliancy, we need a working optimizer.

This section discusses a ranoge of optimizers that might be used, from which it selects the W instance-based planner. The discussion is somewhat technical and may be skippined during a first read of this paper.

[8] X numbers will be ranked $1..n$. Runs of conseutive equal numbers are ranked via their average position. That is, the following $n = 6$ numbers $(1, 2, 2, 3, 4, 5)$ are ranked $1, 2.5, 2.5, 4, 5, 6)$. For more details, see http://goo.gl/FY4Na.

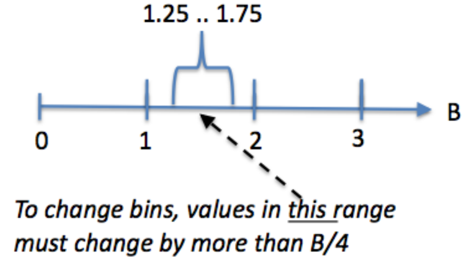


Fig. 3. Ranked values divided into B equal frequency bins. For the majority of the ranking (e.g. those beteen 1.25 and 1.75), any rank change due to noise less than a rank of 0.25 will *not* change the bin of those values.

A. Gradient Descent Optimizers

Using "what-if" queries, it is possible to use software process models to find the fewest changes to a project that most reduce development effort [**?**]. Such optimization tasks are traditionally implemented by computing partial differential equations of a model, and then exploring the surface of steepest change.

A premise of this gradient descent approach is *tuning stability*; i.e. that the gradients at any point in the model can be determined with certainty. Baker [**?**] tested tuning stability in the NASA93 data set (93 NASA projects using the COCOMO ontology, see http://promisedata.org/?p=35). He tuned the COCOMO (a, b) paramets that control for linear and exponential effects (respectively) using from 100 samples of 90% of NASA93. The observed tunings on (a, b) covered a very large range:

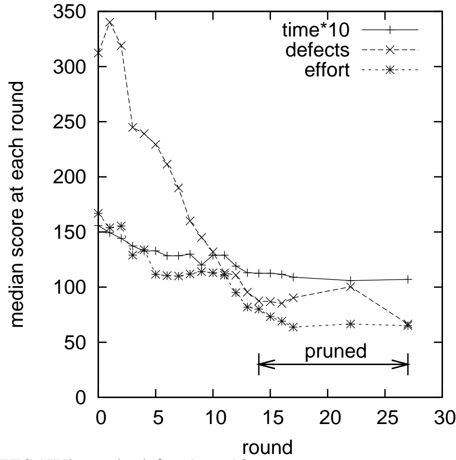
$$(2.2 \leq a \leq 9.18) \wedge (0.88 \leq b \leq 1.09) \quad (1)$$

Gradient descent algorithms are confused by such large amounts of noise. Suppose some proposed technology doubles productivity, but a moves from 9 to 4.5. The improvement resulting from that change would be obscured by tuning instability.

B. AI Optimizers

The previous section argued that traditional gradient descent optimizers executing over simple linear models may be ineffective (due to the noise in SE data). When traditional methods fail, AI search algorithms can sometimes be effective. In this approach, we perturb the internal tunings of a model in accordance with known variations (e.g. the known ranges of the internal COCOMO parameters seen in 30 years of COCOMO reserach). The AI algorithms can then sample the space of possible behaviors, looking for stable conclusions within a large space of possibilities. For example, given a space of *possible* changes to a project, we have used AI search algorithms to find the *smallest* set of changes that *most improves* model output. For example,

- In [**?**], [**?**], we built optimizers using simulated annealling (SA) and COCOMO effort/time/defect models.
- In [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], we showed that SA is outperformed by other AI algorithms. We also found that,



SEESAW's ranked for 1 to 13.

round0: $r_x = \emptyset$	round7: added {rely=3}
round1 added {pmat=3}	round8: added {stor = 3}
round2: added {resl=4}	round9: added {time = 3}
round3: added {team=5}	round10: added {tool = 4}
round4: added {aexp=4}	round11: added {sced = 2}
round5: added {docu=3}	round12: added {site = 4}
round6: added {plex=4}	round13: added {acap = 5}

Fig. 4. Effects of SEESAW optimization. As the x value increases, more and more of SEESAW's recommendations are applied. After some point $x = 13$, future changes to the recommendations result in statistically insignificant changes. From [?].

for the purposes of COOCOMO-based optimization, our own algorithms called SEESAW out-perfored the usual AI algorithms (beam search, A-star, etc) [?].

The output of these AI search algorithms is an ordered set of recommendations. It then takes linear time to explore the first $1 \leq x \leq all$ of these recommendations to find the smallest number of recommendations that most improve a project. For example, in Figure 4, the y-axis shows one case study using the predictions generated by the COCOMO effort/time estimator and the COQUALMO defect estimator[9] The x-axis of that figure shows the effect of set of changes proposed to the project. For example:

- At $x = 0$, we see the predictions from the original project.
- At $x = 1, 2, ..$ etc, we see the results of applying the top ranked, then the second ranked (etc) recommendation generated by the SEESAW optimizer.
- After some point ($x = 13$), the changes no longer become statistically significanly different. After that point, we prune all remaining recommendations.

Observe how the optimization recommendations of SEEAW:

- Dramatically reduces development time (from over 300 calendar months to less than 100);
- Halves the effort prediction (from over 150 months of effort to under 75 months);
- Offers modest reductions in the number of delivered defects (from 150 to 110).

[9] These y-values are the median predictions seen in 100 simulations, where we picked inputs stochastically from the range of known project options.

C. Instance-Based Optimizers

While a successful prototype, SEESAW has certain limitations. In our view, these limitations will be found in any model-based optimizer applied to SE data:

- *Data restrictions:* SEESAW is a wrapper around COCOMO models. Hence, it can only accept projects described in the ontology of that model- a limitation that Shepperd views as a significant drawback in the COCOMO family of models [?].
- *Cost:* SEESAW's optimizations are only as good as the underlying model. Building, tuning, and maintaining convincing software process models is a time consuming tasks. For example, Raffo spent two years building and tuning a software process model of the development processes of a North American software company [?].
- *Over-fitting:* SEESAW samples increasingly narrow segments of the state space of a model ("ying in", as it were, into small cracks between the training data). If the test data does not fall into those tiny region, then the recommendations will fail. For example, when SEESAW's recommendations are tested on project data *not* used to train the model, then those recommendations fail to (e.g.) decrease development effort in over half those experiments [?].

When model-based methods fail, the alternative are instance-based methods that store training data in an n-dimensional space. Inferences are then drawn from the neighors of test instances in the training space.

Instance-based methods use no model, hence they do not demand that the data comes in a particular form. Hence, they do not suffer from *data restrictions*. As to the *cost* issue, they are cheaper to develop and maintain since, in instance-based reasoning, there is no difference in those two activities (when new data arrives, it is added to the training space, thus updating the instance-based reasoner). Finally, regarding *over-fitting*, since there is no model, there no model to overfit.

D. Improving Software Projects with W

W requires a set of historical cases as well as a defined $context$ of controllable feature ranges within those cases. Historical cases are expressed via a set of P project descriptors such as analyst capability, function points, schedule constraints, etc. Each of these cases must also be labed with some known goal metric, such as development effort in man-months. The $context$ consists of a subset of the project descriptors that describe the project at hand, as well as which descriptors are controllable. W uses the entire $context$ to determine the relevant neighborhood, then restricts recommendations to the controllable descriptors only.

For example, a company may track prior software projects via the COCOMO descriptors for analyst capability (acap), schedule requirements (sced), lines of code (kloc), and so on. When planning with W, the project manager could define a new project using all available descriptors. However, business decisions (such as a hiring freeze on analysts) would constrain

the controllable descriptors to only those deemed implementable. From these controllables, W can begin reasoning.

W assumes that a manager can offer us (a) a description of the $context \subseteq P$ that interests them and (b) a list of $control$able options which they can change ($control \subseteq context$). For example, consider the following context for a Finnish banking software project. For simplicity's sake, each descriptor is rated either high, medium, or low:

$$context_1 = \\ ?hw \in \{lo, md\} \wedge ?at \in \{md, hi\} \wedge ?FP \in \{lo, md\} \wedge \\ ?co \in \{lo, md\} \wedge ?prod \in \{md, hi\} \wedge size \in \{md, hi\} \wedge$$

Here, "?" are the $control$ labels; the manager is only concerned with projects that have a low or average number of function points (FP).

W finds a project treatment R_x by studying the project similar to the context in the case repository. Formally, $W2$ explores the $neighorhood$ of the $context$, looking for ways to select for the "best" cases using some utility measure. In this case, minimizing software effort requirements. In W, this is a six step procedure:

1) Divide cases randomly into $train$: $test$ in the ratio $N_1 : N_2$.
2) Use $context$ to find K nearest-neighbors within $train$.
3) Using the utility measure, the neighborhood into (a) the $best$ cases that should be emulated, and (b) the remaining cases to avoid (which we call $rest$).
4) Rank all differences between (a) and (b) according to how strongly they select for the $best$ cases.
5) Use the $train$ set again, experiment with treatments R_x built from the top ranked items found in $Step4$. Return the treatment that selects for the cases in the $train$ set with highest median $value$.
6) Test the treatment from $Step6$ using relevant cases from the $test$ set; i.e. find all rows in the neighborhood of the $context$ in $test$ set; then find the subset of those rows that match the treatment.

The current implementation of W using $N_1 = 2$, $N_1 = 1$, $K = 20$. The following case study demonstrates one run of W using the $Finnish$[10] dataset. After setting aside $2/3$rds of the data for training in $Step1$, Figure 5 shows the most neighborhood generated in $Step2$. Each instance is ranked by how many features fall within $context$, labed as "overlap" in Figure 5. Cells in gray represent ranges that fall outside $context$. The top 20 rows with the highest overlap are selected.

Next, $Step3$ sorts the cases by utility, with $K_1 = 5$ rows placed into the $best$ set and the remaining $K_2 = 15$ rows placed into the $rest$ set (Figure 6). The *contrast set* between these two sets is shown in Figure 7. Frequency counts are computed for each discrete feature value, with each potential treatment R_x ranked as follows:

$$rank(E) * support(E) = \frac{ratio(E|best)^2}{ratio(E|best) + ratio(E|rest)} \quad (2)$$

[10]PROMISE: http://promisedata.org/data

Relevant Set (Training)							
hw	at	FP	co	prod	size	effort	overlap
lo	md	hi	md	md	hi	9.29	5
lo	lo	md	md	hi	md	9.74	5
lo	hi	md	hi	md	md	8.48	5
lo	hi	md	lo	lo	md	7.71	5
lo	md	lo	lo	hi	lo	8.92	5
lo	lo	hi	md	md	hi	9.75	4
md	lo	md	lo	lo	md	7.28	4
lo	lo	md	hi	md	md	9.04	4
lo	lo	md	hi	hi	md	9.08	4
lo	lo	lo	md	hi	lo	8.00	4
lo	lo	lo	md	hi	lo	7.83	4
lo	lo	md	hi	lo	md	7.64	4
lo	lo	hi	lo	lo	hi	8.41	3
hi	lo	hi	lo	md	hi	9.07	3
lo	md	lo	hi	lo	lo	6.38	3
lo	hi	lo	hi	lo	lo	6.13	3
lo	lo	lo	hi	md	lo	7.19	3
lo	lo	lo	lo	lo	lo	7.00	3
lo	lo	lo	hi	md	lo	7.64	2
hi	lo	lo	lo	lo	lo	7.11	2

(other cases omitted)

Fig. 5. 20 most relevant cases in $train$ set for $context_1$.

$Best$ set						
hw	at	FP	co	prod	size	effort
lo	hi	lo	hi	lo	lo	6.13
lo	md	lo	hi	lo	lo	6.38
lo	lo	lo	lo	lo	lo	7.00
hi	lo	lo	lo	lo	lo	7.11
lo	lo	lo	hi	md	lo	7.19

$Rest$ set						
hw	at	FP	co	prod	size	effort
md	lo	md	lo	lo	md	7.28
lo	lo	lo	hi	md	lo	7.64
lo	lo	md	hi	lo	md	7.64
lo	hi	md	lo	lo	md	7.71
lo	lo	lo	md	hi	lo	7.83
lo	lo	lo	md	hi	lo	8.00
lo	lo	hi	lo	lo	hi	8.41
lo	hi	md	hi	md	md	8.48
lo	md	lo	lo	hi	lo	8.92
lo	lo	md	hi	md	md	9.04
hi	lo	hi	lo	md	hi	9.07
lo	lo	md	hi	hi	md	9.08
lo	md	hi	md	md	hi	9.29
lo	lo	md	md	hi	md	9.74
lo	lo	hi	md	md	hi	9.75

Fig. 6. Partitioning $Relevant$ into $Best$ and $Rest$

range	frequency		$b^2/(b+r)$
	b best	r rest	
FP=lo	5/5	4/15	79%
hw=lo	4/5	13/15	38%
co=lo	2/5	5/15	22%
at=md	1/5	2/15	12%
at=hi	1/5	2/15	12%
prod=md	1/5	4/15	9%
co=md	0/5	5/15	0%
prod=hi	0/5	3/15	0%
FP=md	0/5	7/15	0%
hw=md	0/5	1/15	0%

Fig. 7. Controllable features ranked.

Note that all treatments are ranked in Figure 7, even those that never occur in the $best$ set. To decide which treatments should be included as the final recommendation, the best to worst (R_1, R_2, ...) individual treatments are applied to the

train set. Each time a treatment is added, the *train* set is constrained to rows that only match that treatment. The median utility of the constrainted set is compared to the original *train* set. $Step5$ explores R_x upwards from $x = 1$ while:

- The median *value* of the rows selected by R_{x+1} is greater than that of R_x.
- The number of selected rows $|R_x \wedge neighborhood| \geq 3$; In the case of Figure 9, $Step5$ returns R_1 ($FP = lo$).

Figure 9 shows the cases from this neighborhood that satisfy $R_1 : FP = lo$.

Relevant Set (Testing)							
hw	at	FP	co	prod	size	effort	overlap
lo	hi	md	lo	md	md	7.97	6
lo	md	md	md	md	md	8.72	6
lo	md	md	md	hi	md	9.58	6
lo	md	hi	md	hi	hi	9.84	5
lo	md	hi	md	hi	hi	10.04	5
lo	md	md	md	lo	hi	8.45	5
hi	md	**lo**	lo	md	lo	6.67	4
lo	lo	md	lo	lo	md	7.46	4
lo	lo	hi	md	md	hi	8.99	4
lo	md	hi	hi	md	hi	9.83	4
md	lo	hi	lo	lo	hi	8.70	3
hi	lo	**lo**	lo	hi	lo	9.30	3
lo	lo	hi	hi	hi	hi	9.78	3
lo	lo	hi	hi	hi	hi	10.19	3
lo	lo	**lo**	hi	lo	lo	6.36	2

Fig. 8. *Relevant* set (Cases nearest to $context_1$.)

Rows matching $R_1 : FP = lo$						
hw	at	FP	co	prod	size	effort
lo	lo	**lo**	hi	lo	lo	6.36
hi	md	**lo**	lo	md	lo	6.67
hi	lo	**lo**	lo	hi	lo	9.30

Fig. 9. All rows of Figure 12 satisfying $R_1 : FP = lo$.

Figure 9 applies R_1 on the test data. Its impact is reported as the median effort value of the cases. In the case of $FP = lo$, the constrained testing set gives a median effort of 6.67, a 31% reduction from the original estimate of 8.8.

V. RESULTS

We ran W across a series of discretized effort datasets defined in Figure **??**. For bin partitioning we used equal frequency discretization with the number of bins ranging from 2 to 10. All datasets except for the COCOMO-based NASA93 datasets were originally numeric.

As stated earlier, given B' bins, we compute Δ-resiliency by the smallest B' such that optimizations from $B' - 1$ bins are *no worse* than B' and optimizations from $B' + 1$ bins are *no better* than B'. At this point Δ defines the optimizer's resilience for up to $100/B'/4\%$ noise levels.

Figure **??** show the results of this experiement. Note that the leftmost column references the Mann-Whitney U ranked test. Bin sizes of equal rank are defined to be from the same distribution, asserting that performance is statistically *no better* and *no worse*. Consider the Finnish dataset. 4 bin discretization performs as well as 3 bin discretization, but no better than 5

Rank	Dataset	Bins	Median Reduc	Reduction Quartiles 50%
1	china	8Bins	63%	
1	china	6Bins	63%	
1	china	3Bins	60%	
1	china	5Bins	59%	
1	china	10Bins	57%	
1	china	4Bins	55%	
1	china	2Bins	40%	
1	china1	6Bins	70%	
1	china1	4Bins	64%	
1	china1	10Bins	59%	
1	china1	5Bins	54%	
1	china1	2Bins	47%	
1	china1	3Bins	46%	
1	china2	6Bins	83%	
1	china2	5Bins	76%	
1	china2	4Bins	71%	
1	china2	3Bins	61%	
1	china2	10Bins	48%	
1	china2	2Bins	46%	
1	china2	8Bins	42%	
1	kemerer	6Bins	64%	
1	kemerer	4Bins	51%	
1	kemerer	3Bins	49%	
1	kemerer	2Bins	38%	
1	telecom	4Bins	81%	
1	telecom	3Bins	77%	
2	telecom	2Bins	53%	
1	miya	8Bins	72%	
1	miya	2Bins	62%	
1	miya	3Bins	54%	
2	miya	6Bins	40%	
1	finnish	4Bins	24%	
1	finnish	3Bins	22%	
1	finnish	5Bins	20%	
2	finnish	2Bins	19%	
2	finnish	6Bins	17%	

Fig. 10. Results of Discretization on W's effort reduction performace.

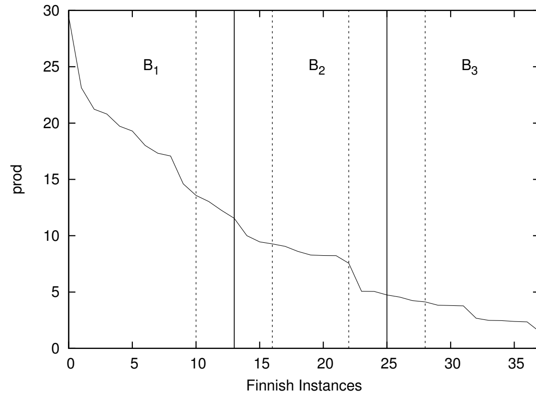
	Rank changes			
Attribute	25%	Median	75%	Δ
hw	0.5	1	1	0
at	1	1	1	1
FP	14	28	64	301
co	1	1	1	2
prod	0.183	0.428	0.874	4.312
lnsize	0.03	0.05	0.115	0.56
effort	0.024	0.0748	0.19	0.5417

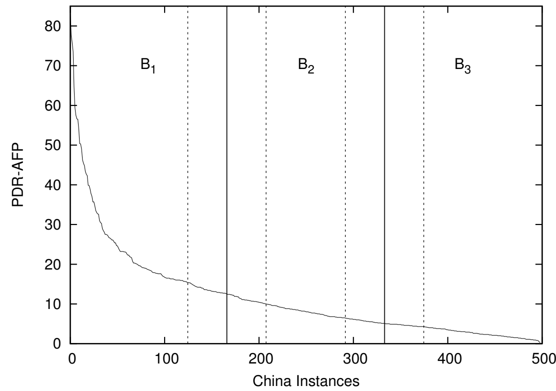
Fig. 11. Measuring Δ-resiliency for the Finnish dataset

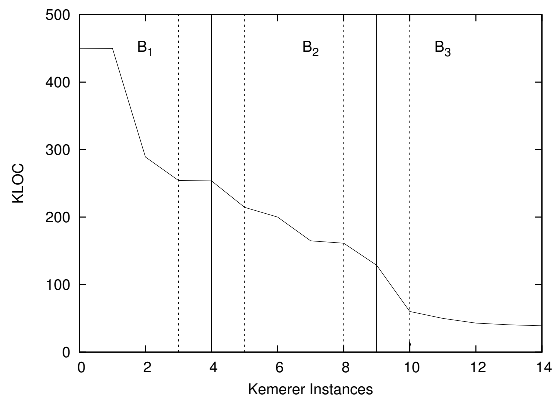
bin discretization. From this we say that $\Delta(finnish|W) = 4$, asserting noise resistance for up to $100/4/4$ = 6.25%.

For other datasets, the smallest Δ-resilient bin size varies between 3 and 4 bins. This may come as a surprise, as Kemerer, Miyazaki, Telecom, China, and Finnish are all continuous datasets with some values carried out to *six* decimal places. Yet, our learner performs just as well when reduced to simple "low, nominal, high" labelling of features.

Overall, our results demonstrate little need for fine granularity in data collection. It's quite possible to reduce data into course chunks, thus making data collection easier.







VI. Determine Local Noise Levels

e.g. do test studies where the same data is collected by N people

VII. External Validity

just for the planning task. further work needed for more tasks.

VIII. Conclusion

The core inution of Δ resilaincy is that data quality is an issue of "fit for purpose". The advantage of this definition is that we can offer a computation definition of data quality (the data has "enough" quality if noise below known levels in the domain do not effect the conclusion). The drawback with this definition is that it is very tasks specific. note that for data warehousing, should not explore just one task. presumably, a warehouse exists for multiple tasks, not all of which may be known at the current time. in this case, the best we can do is determine Δ resiliancy for known tasks, with the aim of increasing our confidence that we have enough noise resilancy for future tasks.

Acknowledgment

	Relevant Set (Testing)															Significant
Data Subset	acap	time	cplx	aexp	virt	data	turn	rely	stor	lexp	pcap	modp	vexp	sced	tool	Cost Drivers
coc81all	o	●	●	●	●	●	●	●	●	●	●	●	●	●	●	15
coc81-mode-embedded	o	●	o	o	●	o	o	o	o	●		●	●	●	●	14
coc81-mode-organic	●	●	o	●	●	●	●		o		●	●	●	●	●	13
nasa93-all	●	●		●	●	●	●	●	●							8
nasa93-mode-embedded	o	●	●		●	●	●	●	●	o	o			●		11
nasa93-mode-semidetached	●			●								o				3
nasa93-fg-ground	●		o	●						●	o					5
nasa93-missionplanning	o	●	●			●	●	●			●	o		o		9
nasa93-avionicsmonitoring	●			o								●	o	o	o	6
nasa93-year-1975	●	●	●	●	●	●		●	●	o	o					10
nasa93-year-1980	●	●	●	o	●	●	●	●	●					●	o	11
nasa93-center2	●	●	●	●	●	o	●	o	●	●	●	●	●		●	14
Usually Significant	5	1	3	5	0	2	2	3	3	3	4	1	2	2	3	
Always Significant	8	11	9	7	11	9	9	8	8	5	4	6	5	5	4	
Total Significant Occurances	13	12	12	12	11	11	11	11	11	8	8	7	7	7	7	

Fig. 12. ●Not significantly different than 10 at 95% Confidence Interval o Not significantly different than 9 or greater at a 95% Confidence Interval