Local vs. Global Models for Effort Estimation and Defect Prediction

Tim Menzies, Andrew Butcher
CS & EE, WVU
Morgantown, USA
tim@menzies.us
abutcher@afrolegs.com

Andrian Marcus
Computer Science,
Wayne State University, USA
amarcus@wayne.edu

Thomas Zimmermann
Microsoft Research
Redmond, USA
tzimmer@microsoft.com

David Cok
GrammaTech Inc.
Ithaca, New York
dcok@grammatech.com

***Abstract*—Data miners can infer rules showing how to improve either (a) the effort estimates of a project or (b) the defect predictions of a software module. Such studies often exhibit *conclusion instability* regarding what is the most effective action for different projects or modules.**

This instability can be explained by data heterogeneity. We show that effort and defect data contain many local regions with markedly different properties to the global space. In other words, what appears to be useful in a global context *is often irrelevant for particular local contexts*.

This result raises questions about the generality of conclusions from empirical SE. At the very least, SE researchers should test if their supposedly general conclusions are valid within subsets of their data. At the very most, empirical SE should become a search for local regions with similar properties (and conclusions should be constrained to just those regions).

***Index Terms*—Data mining, defect/effort estimation, validation, empirical SE.**

I. Introduction

A repeated pattern in software engineering research is *conclusion instability*, i.e. the finding that some effect X is not generally true. For example, in the field of *software development effort estimation*, Mair and Shepperd [1] compared regression to analogy methods. From 20 empirical studies they found no conclusion regarding which methods were best (seven favored regression, four were indifferent and nine favored analogy).

A similar pattern of conclusion instability can be found in *module defect predictors learned from static code features*. Zimmermann et al. [2] learned defect predictors from 622 pairs of projects $\langle project_1, project_2 \rangle$. In only 4% of pairs did defect predictors learned in $project_1$ work in $project_2$. Other conclusion instability results in effort/defect estimation are reported in [1], [3], [4].

The work was partially funded by NSF grant CCF:1017330 and the Qatar/West Virginia University research grant NPRP 09-12-5-2-470. Also, the research reported in this document/presentation was performed in connection with contract/instrument W911QX-10-C-0066 with the U.S. Army Research Laboratory. The views and conclusions contained in this document/presentation are those of the authors and should not be interpreted as presenting the official policies or position, either expressed or implied, of the U.S. Army Research Laboratory or the U.S. Government unless so designated by other authorized documents. Citation of manufacturers or trade names does not constitute an official endorsement or approval of the use thereof. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

One explanation for conclusion instability is *data heterogeneity*. If data sets contain local regions with very different properties, then an induction cross-validation study would exhibit high variance when tests are conducted on stratifications with unusual properties.

To test this conjecture, this paper compares the learning of *treatments* (changes that are intended to improve some quality measure) using:

- Just the *local* data in adjacent clusters; to
- All the *global* data found in all clusters;

Based on the literature on outlier removal in software engineering (e.g. [5], [6]), we expected to find a few heterogeneous clusters spread around other clusters that were mostly homogeneous. To our surprise, we found the reverse. Usually, every cluster was different to the others and in those clusters:

> *The treatments from local regions were different and superior to the global treatments.*

It is neither novel nor interesting to say that particular cases can *sometimes* contradict general principles. What was surprising, however, is that in these results, the global treatments were *nearly always* inferior to the local treatments.

Note that such data heterogeneity explains the prevalence of conclusion instability. It also suggests a change of focus in SE research:

> *Rather than focus on generalities (that may be irrelevant to any particular project), empirical SE should focus more on context-specific principles.*

The rest of this paper is structured as follows. First, we motivate this work with a small case study on local learning. Then, we discuss techniques for localized reasoning:

- The WHERE clustering algorithm that divides the data;
- The WHICH learner that finds treatments in clusters.

We use those techniques in an experiment that compares the treatments learned from global or local contexts. This is followed by notes on validity and related work.

II. Local Learning: Motivations

To motivate this paper, we offer the following simple example of the value of local learning.

There are many general truisms in the field of software engineering. Our results are only interesting if our treatments could

not have been generated in a much simpler way, just by applying the truisms. For example, the COCOMO/COQUALMO effort/defect predictors [7] assume that defects and efforts are mostly reduced by decreasing functionality (measured either in function points or lines of code). According to this assumption, the best thing a manager can do to control defects and cost is to discard needless functionality. This process, of discarding requirements, is also one of Brooks' key recommendations in *The Mythical Man Month* [8].

In the *Experiments* section of this paper, WHICH learns treatments (i.e., changes) from four data sets. Some of those treatments agree with this truism of "make it smaller". For example, WHICH can learn this treatment:

$$loc = 1$$

To read this, note that we discretize variables min..max to 1..7. Hence, $loc = 1$ means "set lines of code to minimum".

Significantly, of the 24 treatments learned in the *Experiments* section, the "make it smaller" treatment appears far more often in the $global$ treatments than in the $local$ treatments (as we expected). Our experiments used four data sets (two on effort estimation and two on defect prediction) and in two of those data sets, the learned $global$ treatment recommends minimizing the function points or lines of code of that system. However, in the 20 $local$ treatments learned by our experiments, only 2 of them recommend "make it smaller". That is, what *seems* to be a good idea overall (e.g. "make it smaller") is actually irrelevant to 18 sub-groups within the data (i.e., local contexts).

Notice that our results do not disagree with Boehm and Brooks. In general, "make it smaller" is a valid method of reducing the effort and defects associated with a software system. Indeed, our $global$ analysis reaches the same conclusion in half the experiments we present below.

However, we would add that for particular kinds of projects, other factors may be more important than just size. For example, for one data set explored below we learn a treatment from a $local$ region of the form

$$pcap = nominal$$

which, in COCOMO-speak, means avoid programmers with poor or very poor programming capability. Note that this treatment makes no reference to the *size* of the system since, this particular data set, the size effects were dominated by the impact of poorly trained programmers. Examples like this motivate our research into local lesson learning in software engineering. Our preferred method of learning those treatments is to combine two tools: WHICH and WHERE.

III. Algorithms: WHICH and WHERE

In order to conduct the experiments of this paper, we need one tool that can learn local lessons from each cluster and a second tool that can find each cluster. This section describe two such tools: the WHICH contrast set learner and the WHERE clusterer.

We currently favor the two tools since they are based on years of our research and incorporate the best practices we have found so far in our work. Also, they scale to large data sets. This does not mean that the community should uncritically accept them. Like any learner, WHERE and WHICH rely on certain tuning parameters to control their operation. We have used our best engineering judgment to set those parameters but it is possible that other settings or, indeed, other algorithms are better suited to this task. A challenge problem we offer other researchers is to review our our methods to propose refinements/alternatives.

But the details of tuning parameters for WHERE and WHICH are orthogonal to this discussion. To defend our conclusions, this paper shows that when the *same analysis method* is applied (1) *globally* to all data or (2) *locally* to just some intra-cluster data, then *different and better treatments* are found from the local analysis. The rest of this paper presents that demonstrations.

A. Contrast Set Learning with WHICH

When we show data mining output to business users, their first question is usually "what does this say about how to improve a project?". To answer this question, we use *contrast set learning* to infer rules describing differences between a current context (called the *baseline*) and a better context (called the *target*). A contrast set rule takes the form

$$if\ R_x\ then\ \ (change = \epsilon_1/\epsilon_0 * support)$$

We say this rule selects some $support\%$ of the data that contains a different (and hopefully better) distribution of the dependent quality variables (and by "select", we mean it finds all rows consistent with R_x). Here, R_x is a $treatment$ containing a set of attribute value pairs a_v; ϵ_0 is the median score of all instances in the baseline and target; and ϵ_1 is the median score in the selected subset of baseline and target. For effort and defect prediction, where *less* is *better*, then the ratio ϵ_1/ϵ_0 is smaller if the treatment selects for *better* instances.

It turns out that the minimal description of the *differences* between two things, is often much smaller than a full *description* of both things. For example, In the experiments shown below, we generate treatments that reference only one attribute. As a result, we can show our users succinct rules describing what needs to change in order to select for certain desired classes.

Our WHICH [9] contrast set learner loops over attribute values combinations, combining those that look most promising:

1) Continuous attributes are discretized to "β" values.
2) A stack is created: one item for every attribute value.
3) The items in that stack are sorted using $\epsilon_1/\epsilon_0 * support$.
4) κ number of times, do:
   - Generate λ number of new items, as follows.
     - Pick two items at random, favoring those with better $\epsilon_1/\epsilon_0 * support$.
     - Combine the pair into a new item. Score it.

technique	name	notes	default
WHICH	β	number of bins for descretizing number attributes	$\beta = 7$ equal frequency bins
	ϵ	generates performance score for a set of instances	e.g. median effort
	λ	number of loops	$\lambda = 5$
	κ	number of pairs to be picked and combined	$\kappa = 20$
	γ	maximum acceptable height of WHICH's stack	$\gamma = \infty$
	ω	minimum acceptable $\epsilon_i/\epsilon_0 * support$ score	$\omega = 0$
WHERE	α	stopping rule for quadtree tree recursion	$\alpha = \sqrt{N}$
	δ	stopping rule for clustering	$\delta = 0.5$

Fig. 1. Default settings (N refers to the number of instances).

- Sort the λ new items rules into the stack.

5) Repeat step 4 until no new improvements seen in the best score. Return the item with best score.

WHICH is controlled by the settings of Figure 1. Two settings change the maximum size of WHICH's stack (γ) and the minimum acceptable $\epsilon_1/\epsilon_0 * support$ score (ω). WHICH runs fastest when γ and ω only allow for small stacks processing rules with largest $\epsilon_1/\epsilon * support$ scores. However, this fast version of WHICH can miss rules which, in isolation, are not promising but, when combined, are useful. To avoid that issue, this study uses $\gamma = \infty$ and $\omega = 0$ (i.e. do not prune the stack and all rules are acceptable).

In the following section, we will use WHICH on pairs of neighboring clusters found by WHERE. In those experiments, where clusters range in size from 20 to 120 instances, WHICH runs very quickly indeed: on a 4GB machine with a 2.5GHz processor, a PYTHON version of WHICH terminates in under a second (excluding time to read any data from disk).

B. Using WHERE to Find Similar Projects

This section describes WHERE, a fast clustering algorithm for finding software artifacts with similar attributes. This process is controlled by the settings of Figure 1.

WHERE clusters data on dimensions synthesized along the axis of greatest variability in the data. One way to find such dimensions is via methods such as principal component analysis (PCA) that transform D basic dimensions (that might be correlated) into a fewer number of uncorrelated (orthogonal) components. In PCA, component I accounts for as much variability as possible in the data and an orthogonal component $I + 1$ tries to account for the remaining.

Matrix factoring methods like PCA take polynomial time to execute [10]. Faloutsos & Lin [11] offer a linear-time heuristic for generating these dimensions. Given N instances, their "FASTMAP" heuristic finds the dimension of greatest variability to a line drawn between the two furthest points. These two points are found in linear time, as follows:

- Pick any instance Z at random;
- Find the instance X that is furthest away from Z;
- Find the instance Y that is furthest away from X;

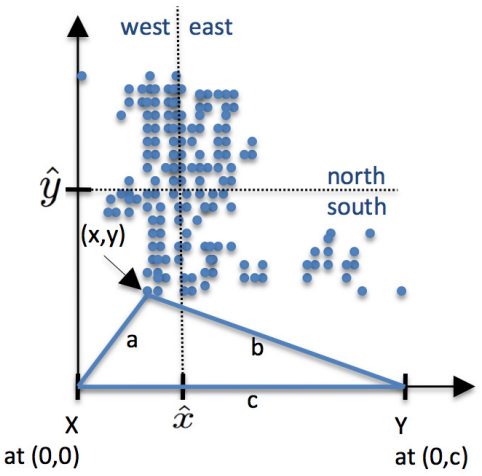

Fig. 2. Each dot is an D-dimensional instance mapped into D=2 using Equation 1. One dimension is the line between X (at the origin) and the most remote instance Y (at $0, c$). Each dot has distance a from the origin and b from the most remote point. The median point on the x and y axis are $\hat{x}$ and $\hat{y}$, respectively. These median points divide the space into four quadrants.

The line $\overline{XY}$ is an approximation to the first component found by PCA and is computed using $2N$ distance calculations (i.e. faster than PCA's polynomial time inference).

As shown in Figure 2, an orthogonal dimension to $\overline{XY}$ can be found by declaring that the line $\overline{XY}$ is of length c and runs from point $(0, 0)$ to $(0, c)$. Each instance now has a distance a to the origin (instance X) and distance b to most remote point (instance Y). From the Pythagoras and cosine rule, each instance is at the point (x, y):

$$\begin{aligned} x &= (a^2 + c^2 - b^2)/(2c) \\ y &= \sqrt{a^2 - x^2} \end{aligned} \quad (1)$$

Figure 2 shows four quadrants defined by the median values of each dimension $(\hat{x}, \hat{y})$: *NorthWest, NorthEast, SouthWest, SouthEast*. WHERE recurses on each quadrant to generate a balanced tree of quadrants (stopping when a sub-quadrant has less than α instances). That is, after an $O(N)$ process that generates the quadtrees, WHERE can use the quadtrees as an index that maps test instances to related instances in time $O(log_4 N)$).

Schikuta [12] warns that quadtrees needlessly sub-divide data when neighboring leaf quadrants have similar properties. Hence, as a post-processor to quadtree generation, WHERE combines similar leaf quadrants as follows:

1) Create a list of leaf quadrants, sorted by their density (number of instances divided by cluster size).
2) Set $stop$ to $\delta *$ maximum density of items in that list.
3) Starting with the densest cluster, perform a geometric search through immediate neighbors of this first quadrant.
4) Remove all quadrants connected in this way from the list and added into their own separate cluster.
5) If the list is not empty, find next cluster (goto step 2).

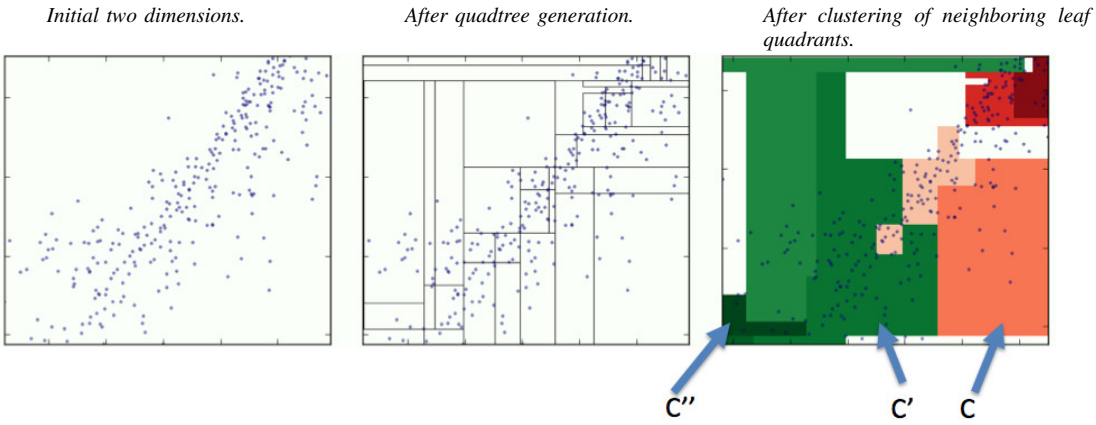

Fig. 3. Recursive dimensionality synthesis. Each 2-d dots represents 24 descriptors of a software project. Colors on the right-hand-side show median intra-cluster development effort (green= lowest effort; red= highest effort). White denotes a region too sparse to cluster. Generated from the NASA93 effort estimation data set (http://goo.gl/WlzCC) using the default parameters of Figure 1.

Figure 3 shows the results of running WHERE on the NASA93 effort estimation dataset from the PROMISE repository (see http://goo.gl/WlzCC). Each dot describes one project using 24 independent attributes and one dependent attribute showing the development effort (in months).

The left-hand-side of that figure places the data within the top two dimensions learned by FASTMAP. The middle figure shows the leaf quadrants found after WHERE recursively explored the NorthWest, NorthEast, SouthWest, SouthEast quadrants. The right-hand-side figure shows the results of leaf quadrant clustering. Each cluster has been colored to show the median intra-cluster development effort. The colors range from dark red (highest effort) to dark green (lowest effort). The white clusters contain less than α members (this occurs when a parent cluster has less than $2 * \alpha$ items).

One inference supported by Figure 3 is *what to change*. Consider the three clusters in Figure 3 labelled C, C', C''. Suppose a manager of a project in the pink cluster C is considering how to decrease the development effort of that project (of all the neighbors of that cluster, the green cluster C' has the lowest development effort). Accordingly, that manager would run WHICH over the C' data to learn treatments that convert projects of type C to C'.

Note that such a strategy is *not* available to the manager projects in the dark green cluster C''. No neighbor of C'' has a shorter development effort so, in that cluster, we would advise to just maintain the status quo.

One advantage of WHERE is that it scales to large data sets. WHERE always recurses on two dimensions synthesized in linear time (via FASTMAP). This approach scales linearly on the number of attributes. WHERE also scales very well on the number of instances. Figure 4 shows runtimes after applying WHERE to data sets from the PROMISE repository (CM1, KB2, MW1, KC3, PC1, KC1 from http://goo.gl/fNgNW) where the instances are copied once, twice, four, or eight times. Note that the runtimes scale linearly with data set size.

One potential drawback with WHERE is that since it uses the FASTMAP heuristic, it may not find the points that best

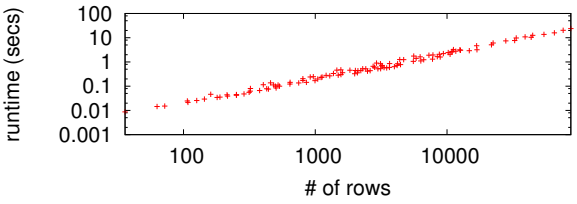

Fig. 4. WHERE's runtimes scale linearly on number of rows. Generated using the defaults on Figure 1.

represent the dimensions of greatest variability in the data. To check this if this heuristic generates inaccurate dimensions, elsewhere [13] we have conducted extensive experiments with the FASTMAP heuristic versus other, more considered clustering methods such as k-means. Our results agree with those of Faloutsos & Lin [11]: in practice, the approximate dimensions found by FASTMAP does not degrade inferencing (compared to other more complete, and slower, approaches).

IV. Experiments

The goal of our experiment is to test *locally* learned treatments are better and different to *global* treatments.

A. Data

This study used data from http://promisedata.org/data:

- CHINA is 499 software development projects, tagged with the development effort (in months). Each project is described in terms of *function points* (i.e. number of high-level operations within the system) shown in Figure 5.
- NasaCoc are the 156 cost estimation instances in the combined NASA93 and COC81 datasets converted to COC-II via Reifer et al.'s RosettaStone algorithm [14][1].
- LUCENE2.4 is a defect log on 340 OO classes in a JAVA search engine optimized for text mining.
- XALAN2.6 is a defect log on the 875 classes of an OO Java implementation of an XLST processor.

[1] http://goo.gl/8hjF5, http://goo.gl/lwxqq

afp	adjusted function points	adjusted size by the standard value adjustment factor (vaf)
input	function points (ufp) of input	
output	function points (ufp) of external output	
enquiry	function points (ufp) of external enquiry	
file	function points (ufp) of internal logical files or entity references	
interface	function points (ufp) of external interface added	function points (ufp) of new or added functions
changed	function points (cfp) of changed functions	
deleted	function points (cfp) of deleted functions	
pdr_ufp	normalized level 1 productivity delivery rate	norm. level 1 effort (for development team) divided by functional size (unadjusted function points).
npdr_afp	normalized productivity delivery rate	normalized effort divided by functional size (unadjusted function points).
npdu_ufp	productivity delivery rate (adjusted function points)	summary work effort divided by adjusted function point count.
resource	team type	1 = development team effort (e.g., project team, project management) ; 2 = development team support (e.g., database administration, data administration, quality assurance) ; 3 = computer operations involvement (e.g., information center support, computer operators, network administration) ' 4 = end users or clients (e.g., user liaisons, user training time)
dev.type	development type	1= new development, 2= enhancement; 3= redevelopment.
duration	total elapsed time for the project	in calendar months.
effort	summary work effort	provides the total effort in hours recorded against the project.

Fig. 5. The function point metrics used in CHINA. The last row is the dependent variable.

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

Fig. 6. The C-K metrics used in LUCENE2.4 and XALAN2.6. The last row is the dependent variable.

The last two data sets describe their classes in terms of the standard C-K metrics for object-oriented systems [15] shown in Figure 6.

These data sets were selected for this study since they are as different as we could find in the PROMISE repository (two relate to effort estimation and the other two to defect prediction; also, CHINA predicts for total staff effort while NasaCoc predict for calendar months of development). Also, they are public domain, so it is possible for other researchers to reproduce, improve, or even refute our conclusions.

Note that WHERE and WHICH can work on defect and effort data (and, indeed, any other source of supervised data) since these techniques make no particular commitment to the semantics of a particular domain.

B. Method

For this study, we used the WHERE and WHICH settings of Figure 1. After WHERE was applied to each data set, then for each cluster C, a *local* analysis was conducted as follows:

- Let M, M' be the median scores of cluster C and neighboring cluster C'. For LUCENE2.4 and XALAN2.6, that score reflects a defect count per class. For the other data set, that score reflects the total staff development effort (CHINA) or the development time (NasaCoc).
- Let N, N' be the number of instances in C, C'. For C' (the neighbor with best M' mean score), use WHICH to learn a *local* treatment (where $\epsilon_0 = M'$ and ϵ_1 comes from the examples selected by the treatment from C').
- This treatment was then tested on C.

Note that cluster C was skipped if it is was too small (for this study, we used $N < 20$) or if there is no neighbor with a better score $M' < M$. In practice, this removed less than 40 instances per data set.

For the *global* analysis, all available training data was sorted on some quality measure. In order to permit valid comparisons, all data from clusters skipped by the *local* analysis was removed. WHICH was used to learn a *global* treatment with $baseline = best \cup rest$ and $target = best$.

cluster	NasaCoc	CHINA	LUCENE2.4	XALAN2.6
global	ltex=1	afp=1	rfc=2	loc=1
C1	pcap=4	added=4	amc=7	amc=1
C2	prec=4	deleted=1	ca=1	cam=2
C3		deleted=1	dam=5	cam=3
C4			mfa=1	dit=2 or 4
C5			moa=1	loc=1
C6				loc =1 or 2
C7				moa=1
C8				moa=4
C9				rfc=1
C10				wmc=3
repeat attribute	0	2 (deleted)	0	3 (loc) 2 (cam) 2 (moa)
repeat attribute values	0	2 (deleted=1)	0	3 (loc=1)
local = global	0	0	0	2 (loc=1)

Fig. 7. Treatments learned by WHICH for clusters found by WHERE.

For both the $local$ and $global$ studies, we reported the distribution of quality measures in (a) the baseline (b) in the instances selected by $local$ and $global$ treatments.

C. Results

Figure 7 shows the attribute values found in the treatments learned from either the $global$ analysis or in a $local$ analysis. All values are discretized to the range 1..7, min..max so, for example, pcap=4 in the NasaCoc results translates to programmer capability is nominal (a.k.a. average) in the standard COCOMO ontology.

Line one of that figure shows the treatments generated via a $global$ analysis. The other lines show treatments learned from local clusters. Different data sets produced differing numbers of clusters: e.g. XALAN2.6 generated ten while NasaCoc generated only two. Of the local treatments generated, none used more than one attribute and only two used more than one value (see *dit* and *loc* in XALAN2.6). For an explanation of the attribute names, see Figure 5 or Figure 6 or [16].

The bottom of Figure 7 shows how often an attribute, or an attribute value, was seen in more than one treatment:

- In two data sets (NasaCoc, LUCENE2.4), all treatments were different.
- In one data set (CHINA) the number of deleted function points was important in two local treatments.
- In the remaining data set, lines of code (loc), cohesion amounts classes (cam), and aggregation (moa) was important in more than one cluster.

Also shown at the bottom of Figure 7 is how often the global and local treatments were the same. This occurred in only one data set (XALAN2.6) and only in two of its ten clusters.

From Figure 7, we can make three important observations:

- *The treatments are succinct:* (never more than one attribute). This is important from a management perspective since it can be difficult for managers to control multiple factors in a project.
- *The local treatments are insightful:* As discussed in §2, defect and effort reduction is often seen as a matter of reducing function points or lines of code. That view can be seen in the $global$ treatments of CHINA and XALAN2.6 that recommends setting function points (afp) and lines of code (loc) to their minimum value. However, in 18 of 20 clusters, such a simplistic recommendation was *not* found in the learned treatments.
- *The local treatments are different to the global treatments:* for 18 out of 20 locally generated treatments.

CHINA (effort)

		treated		
percentile	raw	global	local	
0 th	0	0	0	min = 26
25 th	1	1	1	maxs = 49,034
median = 50 th	4	4	3	
75 th	9	8	7	
worst = 100 th	100	100	74	
75th - 25th	8	8	6	

NasaCoc (months)

		treated		
percentile	raw	global	local(*)	
0 th	0	0	0	min = 4.9
25 th	12	10	2	max = 96.4
median = 50 th	21	17	7	
75 th	30	26	8	
worst = 100 th	100	100	9	
75th - 25th	19	16	6	

LUCENE2.4 (defects)

		treated		
percentile	raw	global	local(*)	
0 th	0	0	0	min = 0
25 th	0	0	0	max = 30
median = 50 th	3	0	0	
75 th	10	3	0	
worst = 100 th	100	23	0	
75th - 25th	10	3	0	

XALAN2.6 (defects)

		treated		
percentile	raw	global	local(*)	
0 th	0	0	0	min = 0
25 th	13	13	13	max = 8
median = 50 th	13	13	13	
75 th	33	25	13	
worst = 100 th	100	100	63	
75th - 25th	20	13	0	

Fig. 8. Effects on effort/defect distributions. All values x normalized via $round(100 * (x - min)/(max - min))$. Raw denotes distributions in original data. $Treated$ denotes distributions in the subset of either (a) the raw data selected by the $global$ treatment or (b) two neighboring clusters selected by the $local$ treatment. $Local(*)$ denotes a distribution than that is statistically different to the $global$ distribution (Mann-Whitney, 95%).

When the treatments of Figure 7 where applied to the data, the distributions of Figure 8 were generated. These distributions are expressed in terms of their percentile bands:

- The *O th* percentile row is the *minimum* observed value;
- The *50 th* percentile row is the *median* observed value.
- The *100 th* percentile row is the *maximal* observed value;

In order to simplify comparisons, all results are normalized 0 to 100 against the minimum to maximum raw value. These raw min and max values are shown on the right-hand-side of each table in Figure 8.

In order to interpret those results, we offer the following notes. In this experiment, an ideal learner:

- Reduces the median defect or costs measures seen in the untreated raw data.

- One treatment learner is superior if the former has a statistically different and lower median than the other.
- Another measure of interest is the *intra-quartile range*; i.e. the 75-25th percentile range: the *smaller* this range, then the *more confidence* we have that the treatment will produce effects around the median value.
- Finally, a treatment learner should not select for outstandingly bad outcomes. Therefore it is useful to consider the *worst-case scenario* seen in the *worst* rows (the 100-th percentile range where defects and effort are maximal).

An examination of the 100-th percentile range shows that the worst-case scenario of *local* learning is much less than other treatments. While this is a clear effect in all the results, it is particularly marked in NasaCoc and LUCENE2.4. In the latter, $local$ treatments avoided *all* defective modules (as witnessed by the column of zeros in the local LUCENE2.4 results).

Another result to note is that (except for CHINA) the spread of the results (75th-25th percentile range) is less with $local$ learning than with $global$. This effect is particularly marked in NasaCoc and XALAN2.6.

In summary, $local$ treatments were different and superior to the $global$ treatments. The differences in those treatments was shown in Figure 7 while the $local$ superiority is shown in Figure 8. In all data sets, $local$ treatments reduced the worst-case scenario and the intra-quartile range. While some of those reductions are modest, others are quite marked. As to the median results, $local$'s medians were never worse than $global$ and, in one case (NasaCoc) they were significantly better.

V. Validity

Assumptions: Our techniques assumed *structure implies behavior* so that changing attribute values will also change the properties of a project (e.g. number of defects or development time). Another assumption made by our techniques is that the dimensions of most interest are the *dimensions of greatest variability*. While we cannot prove these assumptions, we note that they are shared by many other SE researchers:

- Structure implies behavior is a widely-held in the instance-based effort estimation community, where estimates of software effort are generated using nearest neighbor algorithms; e.g. [17]–[21].
- The value of the dimensions of greatest variability is an assumption shared by other researchers such as those using feature weighting based on variance [22] or principal component analysis; e.g. Nagappan, Ball & Zeller [23].

Clustering: One explanation for our experimental results is that our clustering techniques are somehow in error, and that defect/effort estimation should always be based on the groupings identified manually by human experts. We doubt this, for two reasons. Much research concludes that inferencing results improve after dividing data according to automatically inferred clusters: see [24]–[26], and papers at TSE [17], [18], [21] and ASE [20]).

Effects of stochastic search: Another explanation for $local$ being so different to $general$ is that we are sampling both with a stochastic device (recall that WHICH builds rules via a stochastic search through the space old rules; and WHERE builds clusters by picking random points, then searching for points further away from that initial point). Perhaps all that stochastic sampling has added some jitter into the treatment selection? We discount this possibility since we agree with Motwani and Raghavan [27]: when sampling a space containing uncertainty, a randomized optimizer (like WHICH) may give you *greater* stability since it is not distracted by minor gradients in the data. We have seen evidence for this stability in the above results. Recall the intra-quartile ranges of Figure 8: WHICH's rules learned from WHERE's clusters are *more* stable than those learned from the $global$ analysis.

Parameter settings: Finally, the conclusions reached here come from two specific algorithms run under very specific conditions (the parameter settings of Figure 1). Perhaps all our conclusions are due to quirks in those algorithms and settings? This point was discussed at the start of §3. Certainly, there should be more work in the internals of WHERE and WHICH as well as other algorithms that can reason locally within data subsets or globally across all available data. However, the internal details of these techniques is less important than their effects when applied to data. The goal of the above experiments is to check our base premise; i.e. that when the *same* analysis method is applied to *local* and *global* data, then we discover better and different results with the *local* analysis.

VI. Related Work

Issues with Empirical SE: This paper opened with a commentary on conclusion instability in software engineering. Other researchers have made similar comments. Ideally, the practices of software engineers should be based on methods with well-founded support in the literature. Unfortunately, this is not currently possible. In their pessimistically entitled paper "Is Evidence Based Software Engineering mature enough for Practice & Policy?", Budgen et al. [28] warn that the state of the art in empirical SE does not yet recommend itself for setting management policies. Budgen et al.'s solution is to restructure the literature so that it is simpler to search large collections of research papers. We observe that restructuring will not solve the data heterogeneity problem unless researchers stumble on the same clusters found by techniques like WHERE. As shown above, finding those clusters is not a simple manual task. Therefore, we argue for *both* the literature restructuring proposed by Budgen *as well as* the regrouping software artifacts into clusters with similar properties.

Tackling instability: Previously, we have tried to reduce conclusion instability via:

- Feature selection to prune spurious details [29];
- Instance selection to prune irrelevancies [20], [21], [30];
- Extended data collection.
- Monte Carlo simulation over the space of options [31]

Despite all that work, the variance observed in our models remains very large. Even the application of techniques such as instance-based learning have failed to reduce variance in our effort predictions [30]. Feature subset selection has also

been disappointing: while (in our experiments, from 150% to 53% [30], the residual error rates are large enough that it is hard to use the predictions of these models as evidence for the value of some proposed approach. Lastly, further data collection has not proven useful. Certainly, there is an increase in the availability of historical data on prior projects (e.g. in the PROMISE repository used for this study). However, Kitchenham et al. [32] cautions that the literature is contradictory regarding the value of using data from other companies to learn local models.

Context-specific SE: Many authors discuss contextualizing empirical SE. For example, Petersen & Wohlin [33] offer a rich set of dimensions along which software projects can be contextualized (processes, product, organization, market, etc). They offer no way to learn new contextualizations for new projects whereas, in this paper, we only need to run WHERE on new data to find new contexts. Also, while their arguments are convincing, they offer no experimental confirmation that their contexts are the "right" contexts. This paper, on the other hand, offers an operational test for any candidate context: the context is interesting if it results in different and better treatments.

Other clustering techniques: We prefer WHERE to other clustering methods like k-means since WHERE's quadtrees do not require multiple passes to learn the appropriate number of clusters. Also, unlike clustering methods such as EM [34] (that requires some kernel assumptions to define "near" and "far" from a cluster centroid), our cluster algorithm is a non-parametric method that works without kernel tuning. Finally, thanks to leaf quadrant clustering, WHERE can handle clusters of very irregular shapes (while k-means and EM work best on clusters that are mostly convex is shape).

Dimensionality synthesis: In text mining, it is standard practice to infer a reduced set of dimensions via some matrix factorization process such as PCA or the LSI technique preferred by Marcus [35]. Such reduction is essential since, when text mining, the upper bound on the number of dimensions is the number of unique words in a language. Also, some research in effort estimation infers dimensions using PCA as a pre-processor to model construction. For example, Wen et al. use PCA as a pre-processor to analogy based effort estimation [36].

To the best of our knowledge, in the fields of empirical SE, there is no other work combining *both* clustering *and* inferred dimensions. Outside of SE, however, we can find a few examples of such a combined approach. For the purposes of logistic planning, Chen and Meng performing principle components analysis, followed by clustering, as a pre-processor to their planning process [37]. Also, knowledge acquisition researchers also use clustering over synthesized dimensions. For example, in their KSS0 tool, Gaines and Shaw display examples collected from a user in a 2D space defined by the first two dimensions of PCA [38]. This space is then studied to find gaps between the existing instances.

VII. Conclusion and Future Work

It is to be expected that lessons learned *across a population* may be somewhat different to the lessons learned from *individuals within* that population. What was unexpected was just how different where the $local$ treatments, and how important where those differences.

This paper has compared the learning of *treatments* (changes that are intended to improve some quality measure) using:

- Just the *local* data in adjacent clusters; to
- All the *global* data found in all clusters;

After clustering with WHERE, and learning treatments with WHICH, it was found that in 18 out of 20 $local$ treatments, the treatments were completely different to the treatments learned from a $global$ analysis of all the data. When those treatments were applied to the data, distributions were observed with the following properties:

- The $local$ treatments resulted in distributions with lower variance. Specifically, the intra-quartile range was much lower than that seen in the raw data, or the data generated with the $global$ treatments.
- The *worst-case scenario* was much less in the data generated from the $local$ treatments. For example, in one data set (LUCENE2.4) , for defect prediction, the treated $local$ clusters had zero defects (while in the data generated from the $global$ treatments, some defective modules did appear).
- As to the median results, $local$s medians were never worse than the $global$ medians and, in one case (NasaCoc) they were significantly better.

Hence we conclude that the local treatments are both *different and superior to the global treatments*.

This conclusion has implications for the practices and goals of empirical SE. Rather than seek general principles that apply to many projects, we now advise that empirical SE should focus on ways to find the best local lessons for groups of related projects.

Future work should proceed on four fronts:

- Techniques like WHICH and WHERE are useful for finding and exploiting those local groups of related projects. A challenge problem we offer other researchers is to review our our methods to propose refinements and/or alternatives.
- The core experiment of this paper should be repeated on many other data sets. Before making any general pronouncement along the lines of "the best way to improve software developments is...", researchers should check if their preferred method holds for subsets of the data.
- In this paper, we have only explored using WHICH and WHERE to *improve* a project. The conclusions of such an analysis is a recommendation to a manager along the lines of "this is what you should do". An alternate analysis would be to find the actions that most *degrade* a project; e.g. drives projects in cluster C to a *worse* cluster C'. The conclusions of that other kind an analysis would be a recommendation "whatever you do, do <u>not</u> do this".

REFERENCES

[1] C. Mair and M. Shepperd, "The consistency of empirical comparisons of regression and analogy-based software project cost prediction," in *Empirical Software Engineering, 2005. 2005 International Symposium on*, 2005, p. 10 pp.

[2] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction," in *ESEC/FSE'09*, August 2009.

[3] B. Kitchenham, E. Mendes, and G. H. Travassos, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Trans. Softw. Eng.*, vol. 33, no. 5, pp. 316–329, 2007, member-Kitchenham, Barbara A.

[4] B. Turhan, T. Menzies, A. Bener, and J. Distefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 68, no. 2, pp. 278–290, 2009, available from http://menzies.us/pdf/08ccwc.pdf.

[5] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *ICSE'11*, 2011.

[6] K.-A. Yoon and D.-H. Bae, "A pattern-based outlier detection method identifying abnormal attributes in software project data," *Information and Software Technology*, vol. 52, no. 2, pp. 137 – 151, 2010.

[7] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts, *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[8] F. P. Brooks, *The Mythical Man-Month, Anniversary edition*. Addison-Wesley, 1995.

[9] L. Huang, D. Port, L. Wang, T. Xie, and T. Menzies, "Text mining in supporting software systems risk assurance," in *IEEE ASE'10*, 2010, pp. 163–166, available from http://menzies.us/pdf/10textrisk.pdf.

[10] Q. Du and J. E. Fowler, "Low-Complexity Principal Component Analysis for Hyperspectral Image Compression," *International Journal of High Performance Computing Applications*, vol. 22, no. 4, pp. 438–448, Nov. 2008.

[11] C. Faloutsos and K.-I. Lin, "Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '95. New York, NY, USA: ACM, 1995, pp. 163–174. [Online]. Available: http://doi.acm.org/10.1145/223784.223812

[12] E. Schikuta, "Grid-clustering: A hierarchical clustering method for very large data sets," in *In Proceedings 15th Joint. Conf. on Pattern Recognition*, 1993, pp. 101–105.

[13] G. G. Adam Nelson, Tim Menzies, "Sharing experiments using open source software," *Software- Practice and Experience*, vol. 41, no. 3, pp. 283–305, March 2011, available from http://menzies.us/pdf/10ourmine.pdf.

[14] S. C. D. Reifer, B. Boehm, "The rosetta stone: Making cocomo 81 estimates work with cocomo ii," *Crosstalk*, pp. 11–15, Feb 1999, available from http://goo.gl/f53OL.

[15] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, pp. 476–493, 1994.

[16] B. Boehm, "Safe and simple software cost analysis," *IEEE Software*, pp. 14–17, September/October 2000, available from http://www.computer.org/certification/beta/Boehm_Safe.pdf.

[17] J. W. Keung, B. A. Kitchenham, and D. R. Jeffery, "Analogy-x: Providing statistical inference to analogy-based software cost estimation," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 471–484, 2008.

[18] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering*, vol. 23, no. 12, November 1997, available from http://www.utdallas.edu/~rbanker/SE_XII.pdf.

[19] F. Walkerden and R. Jeffery, "An empirical study of analogy-based software effort estimation," *Empirical Softw. Engg.*, vol. 4, no. 2, pp. 135–158, 1999.

[20] E. Kocaguneli, G. Gay, T. Menzies, Y. Yang, and J. W. Keung, "When to use data from other projects for effort estimation," in *IEEE ASE'10*, 2010, available from http://menzies.us/pdf/10other.pdf.

[21] E. Kocaguneli, T. Menzies, A. Bener, and J. Keung, "Exploiting the essential assumptions of analogy-based effort estimation," *IEEE Trans. SE (pre-print)*, 2011.

[22] D. Baker, "A hybrid approach to expert and model-based effort estimation," Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007, available from https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443.

[23] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06, 2006, pp. 452–461.

[24] E. Frank, M. Hall, and B. Pfahringer, "Locally weighted naive bayes," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 2003, pp. 249–256.

[25] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid," in *Knowledge Discovery and Data Mining*, 1996, pp. 202–207.

[26] B. Turhan, A. Bener, and T. Menzies, "Nearest neighbor sampling for cross company defect predictors," in *Proceedings, DEFECTS 2008*, 2008, hW.

[27] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995, (reprinted 1997,2000).

[28] B. K. D. Budgen, P. Brereton, "Is evidence based software engineering mature enough for practice & policy?" in *33rd Annual IEEE Software Engineering Workshop 2009 (SEW-33), Skvde, Sweden*, 2009.

[29] Z. Chen, T. Menzies, and D. Port, "Feature subset selection can improve software cost estimation," in *PROMISE'05*, 2005, available from http://menzies.us/pdf/05/fsscocomo.pdf.

[30] T. Menzies, Z. Chen, J. Hihn, and K. Lum, "Selecting Best Practices for Effort Estimation," *IEEE Transactions on Software Engineering*, vol. 32, pp. 883–895, 2006.

[31] P. Green, T. Menzies, S. Williams, and O. El-waras, "Understanding the value of software engineering technologies," in *IEEE ASE'09*, 2009, available from http://menzies.us/pdf/09value.pdf.

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

[33] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, Oct. 2009, pp. 401 –404.

[34] I. H. Witten, E. Frank, and M. Hall, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2011.

[35] D. Poshyvanyk and A. Marcus, "Combining formal concept analysis with information retrieval for concept location in source code," in *Program Comprehension, 2007. ICPC '07. 15th IEEE International Conference on*, june 2007, pp. 37 –48.

[36] J. Wen, S. Li, and L. Tang, "Improve analogy-based software effort estimation using principal components analysis and correlation weighting," in *Software Engineering Conference, 2009. APSEC '09. Asia-Pacific*, dec. 2009, pp. 179 –186.

[37] Z. xia Chen and Q. yong Meng, "Principal component analysis and cluster analysis in multi-regional logistics planning–taking zhejiang province for example," in *Business and Information Management, 2008. ISBIM '08. International Seminar on*, vol. 2, dec. 2008, pp. 15 –18.

[38] B. Gaines and M. Shaw, "Eliciting knowledge and transferring it effectively to a knowledge-based system," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 5, no. 1, pp. 4 –14, feb 1993.