Cross- vs Within-Company Defect Prediction Studies

Tim Menzies, *Member, IEEE,* Burak Turhan, Justin Distefano, and Ayes Bener

***Abstract*—**
OBJECTIVE: The objective of this paper is to determiner under what circumstances individual organizations would be able to rely on cross-company (CC) or within-company (WC)-based defect prediction.

METHOD: Detectors predictors were trained from (a) eleven sets, then tested on the twelth; or (b) trained and tested on the same data set; or (c) trained and tested on a decreasing amount of within-company data.

RESULTS: Using within-company data, adequate defect predictors can be learned from as few as two dozen faulty modules. However, in $\frac{8}{12}$ data sets, large amounts of cross-company data significantly increased *both* the median proablities of pd detection and pf false alarms: $pd/pf = 80/26\%$ (WC) and $91/53\%$ (CC).

CONCLUSION: Companies lacking local data can use cross-company data to predict detects. Given enough cross-company data, such predictors will have large false alarm rates but a much larger probability of detection. Cross-company learning is hence indicated for mission critical software were the extra costs associated with high false alarm rates are compensated by the associated increase in software assurance. For other classes of software, false alarm rates can be decreased using a small amount of local data. In our experiments, the use of local data halved the false alarm rate while only decreasing pd by $\approx 10\%$.

I. Introduction

Software managers can use predictor of defective modules to focus the testing on parts of the system that seem defect-prone. These potential defect-prone trouble spots can then be examined in more detail by, say, model checking, intensive testing, etc. Previously we have explored the use of data miners to learn such predictors from static code attributes. We found that a large class of predictors proposed in numerous publications were out-performed by Bayes classifiers that combined partial evidence from multiple attributes [1].

Kitchenham et.al. would characterize that previous work as a *within-company* (WC) study where the data from one organization is applied to new examples within the same organization. They comment that such WC studes have certain backbacks:

- "The time required to accumulate enough data on past projects from a single company may be prohibitive.
- "By the time the data set is large enough to be of use, technologies used by the company may have changed, and older projects may no longer be representative of current practices."

Therefore, companies are tempted to eskew WC studies, preferring instead to use some cross-company (CC) study where data from elsewehre is applied locally. The value of cross- vs within-company studies for effort estimation has been the subject of much recent work. Kitchenham et.al. [2] conclude that the value of CC vs WC data is unclear.

> *Based on our review it is clear that some organisations would benefit from using models derived from cross-company benchmarking databases but others would not.* [2].

For example, Mendes et.al. [3] found within-company data that performed much better than cross-company data for predicting estimation effort of web-based projects. They only recommend using cross-company data in the special case when that "data is obtained using rigorous quality control procedures".

In the special case of effort prediction from static code features, some of those quality controls can be automated. In the following cross- vs within- study, features of code modules were extracted automatically using the same tool set (the McCabe HiQ tool[1]) that is widely used in industry[2]. We find that

A. Detector Mathematics

Let $\{A, B, C, D\}$ denote the true negatives, false negatives, false positives, and true positives (respectively) found by a binary detector. Certain standard measures can be computed from A, B, C, D:

$$\begin{array}{ll} pd = recall = & \frac{D}{B+D} \\ pf = & \frac{C}{A+C} \\ prec = precision = & \frac{D}{D+C} \\ acc = accuracy = & \frac{A+D}{A+B+C+D} \\ selectivity = & \frac{C+D}{A+B+C+D} \\ neg/pos = & \frac{A+C}{B+D} \end{array}$$

Dr. Menzies and Mr. Distefano are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University. Mr. Distefano is also Chief Programmer and Integrated Software Metrics. Emails: tim@menzies.us, jdistefano@ismwv.com.

Mr. Turhan and Dr. Bener are with the Department of Computer Engineering, Bogazici University, Turkey. Emails: turhanbboun.edu.tr, bener@boun.edu.tr.

The research described in this paper was supported by Bogazici University research fund under grant number BAP-06HA104 and at West Virginia University under grants with NASA's Software Assurance Research Program. Reference herein to any specific commercial product, process, or service by trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

See See http://menzies.us/pdf/07interra.pdf for an earlier draft of this paper. Manuscript received Ocober 1, 2007; revised XXX,

[1] justin?

[2] jsutin- can we say that predict is an alternate data collector that HiQ

The last measure (neg/pos) is quite interesting and can be used within the above equations as follows:

$$prec = \frac{D}{D+C} = \frac{1}{1+\frac{C}{D}} = \frac{1}{1+neg/pos \cdot pf/recall} \quad (1)$$

Ideally, our detectors have high precision *and* high recall *and* low false alarm rates. This ideal situation is very hard to achieve. In practice, detectors must trade-off one goal for the other.

B. Trade-offs with Recall (pd) and False Alarm Rates (pd)

Detector performance can be assessed via *receiver-operator* (ROC) curves. Formally, a defect *predictor* hunts for a *signal* that a software module is defect prone. Signal detection theory [4] offers ROC curves as an analysis method for assessing different predictors. A typical ROC curve is shown in Figure 1. The y-axis shows probability of detection (pd) and the x-axis shows probability of false alarms (pf). By definition, the ROC curve must pass through the points $pf = pd = 0$ and $pf = pd = 1$ (a predictor that never triggers never makes false alarms; a predictor that always triggers always generates false alarms). Three interesting trajectories connect these points:

1) A straight line from (0,0) to (1,1) is of little interest since it offers *no information*: i.e. the probability of a predictor firing is the same as it being silent.
2) Another trajectory is the *negative curve* that bends away from the ideal point. Elsewhere [5], we have found that if predictors negate their tests, the negative curve will transpose into a *preferred curve*.
3) The point ($pf = 0$, $pd = 1$) is the ideal position (a.k.a. "sweet spot") on a ROC curve. This is where we recognize all errors and never make mistakes. *Preferred curves* bend up towards this ideal point.

In the ideal case, a predictor has a high probability of detecting a genuine fault (pd) and a very low probability of false alarm (pf). This ideal case is very rare. The only way to achieve high probabilities of detection is to trigger the predictor more often. This, in turn, incurs the cost of more false alarms.

In practice, engineers *balance* between pf and pd. To operationalize this notion of *balance*, we define bal to be the Euclidean distance from the sweet spot $pf = 0, pd = 1$ to a pair of $< pf, pd >$. For convenience, we (a) normalize bal by the maximum possible distance across the ROC square ($\sqrt{2}$); (b) subtract this from 1; and (c) express it as a percentage; i.e.

$$balance = bal = 1 - \frac{\sqrt{(0-pf)^2 + (1-pd)^2}}{\sqrt{2}} \quad (2)$$

Hence, better and *higher* balances fall *closer* to the desired sweet spot of $pf = 0, pd = 1$.

C. Trade-offs with Precision

Note that Equation 1 can be re-expressed as follows:

$$pf = \frac{pos}{neg} \cdot \frac{(1-prec)}{prec} \cdot recall \quad (3)$$

That is, in Equation 3, when $recall$ is fixed then false alarm rate becomes controlled by precision and a fixed constant determined by the data set being examined; i.e. when ($\alpha = neg/pos$) and $recall = 1$ then:

$$pf = \alpha \cdot \frac{1-prec}{prec} \quad (4)$$

From Equation 4, it is clear that it is porblematic to demand detectors with high recall *and* high precision *and* low false alarm rates. For example, for any targeted recall value, *increasing* precision requires *decreasing* false alarm rates; e.g. for $prec \in \{0.5, 0.70, 0.9, 0.95\}$, pf becomes $\{1, 0.43, 0.11, 0.005\}$, respectively. The effect is particularly marked for data sets with large neg/pos ratios.

Many software engineering data sets have extremely large neg/pos ratios. For example:

- In our previous DMP study [6], the data sets studied had neg/pos ratios of 1.04, 7.33, 9, 10.11, 13.29, 15.67, and 249.
- Hayes, Dekhtyar and Sundaram [7] use text mining to find pairs of connected requirements in a corpus of 220 requirements and 235 design elements (the same CM-1 dataset used in [6]). The total number of possible links in the dataset is $220 \cdot 235 = 51,700$, while the the ground truth RTM contains 361 links, for the neg/pos ratio of $51,700/361 = 143.2$.
- For an extreme example, Google reports that over 10^9 web pages contain the phrase "software" but only one them is the home page of this journal. Hence, neg/pos for web searching is at least 10^9.

Figure 2 graphs Equation 1 for the DMP neg/pos ratios. Figure 3 does the same, but is restricted to zones of higher precision: only the surface for $0.5 \leq prec \leq 1$ is shown. That shadow of the surface on the bottom plane shows that this zone of high precision, high recall, and large neg/pos. As neg/pos increases, high recall&precision is only possible when pf becomes vanishingly small. For example, in the ranges $0.65 \leq prec, recall \leq 0.8$, Equation 3 reports that pf falls into the following ranges:

- $0.023 \leq pf \leq 0.062$ for $neg/pos = 7$;
- $0.0108 \leq pf \leq 0.0287$ for $neg/pos = 15$;
- $0.007 \leq pf \leq 0.0017$ for $neg/pos = 250$;

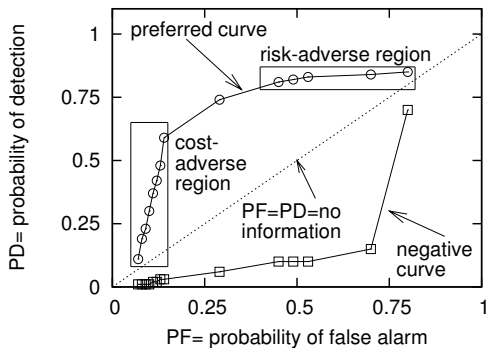


Fig. 1. Regions of a typical ROC curve.

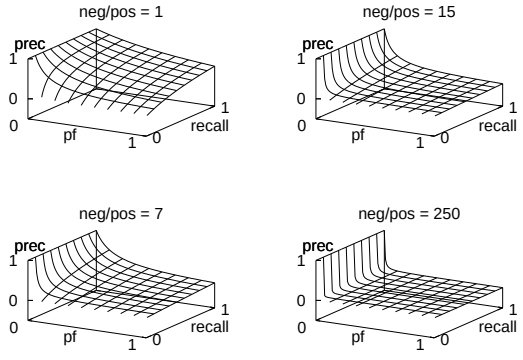


Fig. 2. The relationship between $pd, prec, recall$ and neg/pos.

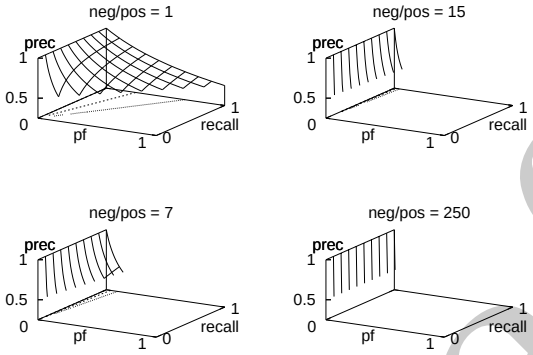


Fig. 3. Figure 2, cropped to the region where $prec > 0.5$.

Detectors learned in the domain of software engineering rarely yield high precision detectors (see Figure 4). Using the above equations, the reasons for this are very clear:

- Those detectors all try to maximize recall;
- Figure 2 shows that such detectors can only achieve high precision in the rare case of very low pf.

Not only do our equation explain the Figure 4 results, they also inform the the instability of precision *and* the stability of pf and pd (recall) seen in previous studies [8]"

- Note how, in Figure 2, that at very small pf values, tiny changes in pf can lead to very large changes in $prec$ (sudden jumps from zero to one).
- The other measures in Figure 2, on the other hand, change far more smoothly and slowly.

That is, the mathematics of these detectors explains numerous prior results such as those shown in Figure 4 (i.e. [6], [7], [9], [10], [11], [12], [13]).

1) When Low Precision is Useful: Achieving both high precision& recall & low false alarm rates can be problematic. Optimizing for one often compromises the other (especially for data sets with large neg/pos ratios). Fortunately, there are may industrial situations where low precision and high recall detectors are useful. For example, one of us (DiStefano) has used our low-precision detectors to review flight code developed at the NASA Glenn Research Center (Ohio, USA). When the results of these detectors were presented to the lead flight engineer, he confirmed that the identified sections (which did not have any recorded defects) had been problematic to maintain, and contained several bugs which had not yet been entered into the defect system.

For another example, from outside the field of SE, a user of commercial web search engine like Google can quickly flick through (say) three pages of results before finding a page of interest. Google has so many return customers since even with precisions of (say) $\frac{1}{30}$, the effort involved in looking at a page is so low that users don't mind examining 29 false alarms.

More generally, there are several situations where low precision detectors are useful:

- *When the cost of missing the target is prohibitively expensive.* In mission critical or security applications, the goal of 100% recall may be demanded in all situations, regardless of the precision.
- *When only a small fraction the data is returned.* Hayes, Dekhytar, & Sundaram call this fraction $selectivity$ and offer an extensive discussion of the merits of this measure [7].
- *When there is little or no cost in checking false alarms.* For example, a detector we have found useful in industrial settings is to check modules where

$$\frac{lines\ of\ comments}{lines\ of\ code} > 0.25$$

This detector triggers on complex functions that programmers comment extensively, instead of splitting up into smaller, more maintainable, functions. This detector is imprecise- it often triggers on well-written functions with detailed comments. However, based on commercial experience, we assert that it is fast and simple for a human agent to inspect the identified modules and discern which

- The DMP paper achieved recalls over 70% and minimum false alarm rates of 15%. Using our equation and data from DMP, it can be shown that the DMP precisions were quite low: $\{min, median, max\} = \{2\%20\%, 70\%\}$ (the last number came from a data set with $neg/pos \approx 1$).
- Huang et.al. [10] won "best paper" at the 2006 IEEE Requirements Engineering conference with detectors exhibiting $prec \approx 0.25$.
- Without extensive feedback from human experts, the classifiers used by Hayes, Dekhtyar, and Sundaram exhibited $pf > 0.6$ (hence, very low precisions) [7].
- Other such as Antoniol et.al. [11], [12] and Marcus et.al. [13] researchers into software tracability report that high recall is achievable only with low precision detectors. Antoniol has also been exploring bug traces in Mozilla components and has found the same high-precision, low-recall trade-off.
- After much experiments with linear regression, model tree learners, Bayes classifiers, decision tree learners, singleton rule learners, and some home-brew learners [9], the general trend is very clear. For those learners, executing on the DMP datasets, $pf \leq pd - 0.5$; That is, for those learners and those data sets, obtaining high recall figures of $pd > 0.6$ implies $pf > 0.1$ and, consequently, low precision.

Fig. 4. Some low precisions seen in the SE literature.

ones were well-written and which were over-commented to disguise being badly coded. We use this detector to find code that should be rewritten prior to release.

II. DATA

This section discusses the data used in this study. Each module of each data sets describes the attributes of that module, plus the number of defects known for that module. This data comes from eight sub-systems taken from four systems. These systems were developed in different geographical locations across North America. Within a system, the sub-systems shared some a common code base but did not pass personnel or code between sub-systems.

Each data set was pre-processed by removing the module identifier attribute (which is different for each row). Also, the $error\_count$ column was converted into a boolean attribute called *defective?* as follows: $defective? = (error\_count \geq 1)$. Finally, the $error\_density$ column was removed (since it can be derived from line counts and $error\_count$). The pre-processed data sets had 38 attributes, plus one target attribute (*defective?*) shown in Figure 5 and include Halstead, McCabe, lines of code, and other miscellaneous attributes.

m = Mccabe		$v(g)$	cyclomatic_complexity
		$iv(G)$	design_complexity
		$ev(G)$	essential_complexity
locs	loc		loc_total (one line = one count
	loc(other)		loc_blank
			loc_code_and_comment
			loc_comments
			loc_executable
			number_of_lines (opening to closing brackets)
Halstead	h	N_1	num_operators
		N_2	num_operands
		μ_1	num_unique_operators
		μ_2	num_unique_operands
	H	N	length: $N = N_1 + N_2$
		V	volume: $V = N * log_2 \mu$
		L	level: $L = V^*/V$ where $V^* = (2 + \mu_2{}^*)log_2(2 + \mu_2{}^*)$
		D	difficulty: $D = 1/L$
		I	content: $I = \hat{L} * V$ where $\hat{L} = \frac{2}{\mu_1} * \frac{\mu_2}{N_2}$
		E	effort: $E = V/\hat{L}$
		B	error_est
		T	prog_time: $T = E/18$ seconds
misc = Miscellaneous			branch_count
			call_pairs
			condition_count
			decision_count
			decision_density
			design_density
			edge_count
			global_data_complexity
			global_data_density
			maintenance_severity
			modified_condition_count
			multiple_condition_count
			node_count
			normalized_cyclomatic_complexity
			parameter_count
			pathological_complexity
			percent_comments

Fig. 5. Attributes used in this study.

The Halstead attributes were derived by Maurice Halstead in 1977. He argued that modules that are hard to read are more likely to be fault prone [14]. Halstead estimates reading complexity by counting the number of operators and operands in a module: see the h attributes of Figure 5. These three raw h Halstead attributes were then used to compute the H: the eight derived Halstead attributes using the equations shown in Figure 5. In between the raw and derived Halstead attributes are certain intermediaries (which do not appear in the MDP data sets);

- $\mu = \mu_1 + \mu_2$;
- minimum operator count: $\mu_1^* = 2$;
- μ_2^* is the minimum operand count and equals the number of module parameters.

An alternative to the Halstead attributes are the complexity attributes proposed by Thomas McCabe in 1976. Unlike Halstead, McCabe argued that the complexity of pathways *between* module symbols are more insightful than just a count of the symbols [15]. The first three lines of Figure 5 shows McCabe three main attributes for this pathway complexity. These are defined as follows. A module is said to have a *flow graph*; i.e. a directed graph where each node corresponds to a program statement, and each arc indicates the flow of control from one statement to another. The *cyclomatic complexity* of a module is $v(G) = e - n + 2$ where G is a program's flow graph, e is the number of arcs in the flow graph, and n is the number of nodes in the flow graph [16]. The *essential complexity*, $(ev(G))$ or a module is the extent to which a flow graph can be "reduced" by decomposing all the subflowgraphs of G that are *D-structured primes* (also sometimes referred to as "proper one-entry one-exit subflowgraphs" [16]). $ev(G) = v(G) - m$ where m is the number of subflowgraphs of G that are D-structured primes [16]. Finally, the *design complexity* $(iv(G))$ of a module is the cyclomatic complexity of a module's reduced flow graph.

At the end of Figure 5 are a set of *misc* attributes that are less well-defined than lines of code attributes or the Halstead and McCabe attributes. The meaning of these attributes is poorly documented in the MDP database. Indeed, they seem to be values generated from some unknown tool set that was available at the time of uploading the data into the MDP. Since there are difficulties in reproducing these attributes at other sites, an argument could be made for removing them from this study. A counter-argument is that if static code attributes are as weak as suggested by Shepherd, Ince, Fenton and Pfleeger then we should use all possible attributes in order to make maximum use of the available information. This study took a middle ground: all these attributes were passed to the learners and they determined which ones had most information.

An interesting repeated pattern in our data sets are *exponential distributions* in the numeric attributes. Elsewhere, we have conducted limited experiments suggesting that a *logarithmic filter* on all numeric values might improve predictor performance [5]. Such a filter replaces all numerics n with their logarithm. $ln(n)$. The log-filtered values are now more evenly spread across the y-range, making it easier to reason about them. To test the value of log-filtering, all the data was

passed through one of two filters: Hence, all our numerics were pre-processed as follows:

- *logNums*; i.e. logarithmic filtering where all numbers n where replaced with $log(n)$.
- To avoid numerical errors with $ln(0)$, all numbers under 0.000001 are replaced with $ln(0.000001)$.

III. Non-parametric Analysis of Results

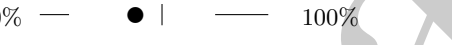
A. Quartile Charts of Performance Deltas

The experiment described below explored numerous combinations of filters, attributes, and learners all within a $(M = 20) * (N = 10)$-way cross-evaluation study. Hence, this experiment generated nearly thousands of measuring pd, pf, bal values from different *methods* applied to the same $test$ data.

The performance results for each method were sorted and displayed as *quartile charts*. To generate these charts, the performance deltas for some $method$ were sorted to isolate the median and the lower and upper quartile of numbers.

$$\{\overbrace{4,7,15}^{q1},20,31,\underbrace{40}_{median},52,64,\overbrace{70,81,90}^{q4}\}$$

In a quartile chart, the upper and lower quartiles are marked with black lines; the median is marked with a black dot; and vertical bars are added to mark the zero value. The above numbers would therefore be drawn as follows:



We prefer quartile charts of performance deltas to other summarization methods for M*N studies. Firstly, they offer a very succinct summary of a large number of experiments. For example, in the next few pages we will display thousands of results in very little area. Secondly, they are a *non-parametric* display; i.e. they make no assumptions about the underling distribution. Standard practice in data mining is to compare the mean performance of different methods using t-test [17]. T-tests are a *parametric method* that assume that the underling population distribution is a Gaussian. Recent results suggest that there are many statistical issues left to explore regarding how to best to apply those t-tests for summarizing M*N-way studies [18], [19].

B. Comparison Statistics

The problem of comparatively assessing L learners run on multiple sub-samples of D data sets has been extensively studied in the data mining community. T-tests that assume Gaussian distributions are strongly deprecated. For example, Demsar [19] argues that non-Gaussian populations are common enough to require a methodological change in data mining.

After reviewing a wide range of comparisons methods[3], Demsar advocates the use of the 1945 Wilcoxon [20] signed-rank test that compares the ranks for the positive and negative differences (ignoring the signs). Writing five years early, Kitchenham et.al. [21] comment that the Wilcoxon test has its limitations. Demsar's report offers the same conclusion, noting that the Wilcoxon test requires that the sample sizes are the same. To fix this problem, Demsar augments Wilcoxon with the Friedman test.

One test not studied by Demsar is Mann and Whitney's 1947 modification [22] to Wilcoxon rank-sum test (proposed along with his signed-rank test). We prefer this test since:

- The Mann-Whitney U-test does not require that the sample sizes are the same. So, in a single U-test, learner L_1 can be compared to all its rivals.
- The U-test does not require any post-processing (such as the Friedman test) to conclude if the median rank of one population (say, the L_1 results) is greater than, equal to, or less than the median rank of another (say, the $L_2, L_3, .., L_x$ results).

The U-test uses *ranks*, not precise numeric values. For example, if treatment A generates $N_1 = 5$ values {5,7,2,0,4} and treatment B generates $N_2 = 6$ values {4,8,2,3,6,7}, then these sort as follows:

Samples	A	A	B	B	A	B	A	B	A	B	B
Values	0	2	2	3	4	4	5	6	7	7	8

On ranking, averages are used when values are the same:

Samples	A	A	B	B	A	B	A	B	A	B	B
Values	0	2	2	3	4	4	5	6	7	7	8
Ranks	1	2.5	2.5	4	5.5	5.5	7	8	9.5	9.5	11

Note that, when ranked in this manner, the largest value (8) gets the same rank even if it was ten to a hundred times larger. That is, such rank tests are less susceptible to large outliers.

Figure 6 shows the U-test for A and B. The test concludes that these treatments have the same median ranked values (at the 95% significance level). As defined in Figure 6, this test counts the $wins$, $ties$, and $losses$ for A and B (where A and B are single or groups of methods). Since we seek methods that can be rejected, the value of interest to us is how often methods *lose*.

IV. Intra-vs-Inter Results

In this first round of experiments, we explored using data from local or imported sources. In each case, some data set was declared "local" and another was declared "import" . A training set was drawn from "import" and applied to a test set created from 10% drawn from "local" (using stratified sampling).

In the "intra" study, the "local" data set was the same as "import". That is, in these intra-experiments, detectors were learned and tested from the same data set.

In the "inter" study, the "local" data set was different to "import". To be precise, "import" was all data from all data set except the "local" data. That is, in these inter-experiments, detectors were learned from all avilable data, except data from the test "local" set.

Measured in terms of pd and pf, the main effect seen in 8 test sets was that importing lots of data (in the inter-experiment) increased *both* pd and pf (see group "D", below). In three other data sets (see groups "B" and "C") the changes

[3]Paired t-tests with and without the use of geometric means of the relative ratios; binomial tests with the Bonferroni correction; paired t-tests; ANOVA; Wilcoxon; Friedman

The sum and median of A's ranks is

$$\begin{array}{rcl} sum_A & = & 1+2.5+5.5+7+9.5=25.5 \\ median_A & = & 5.5 \end{array}$$

and the sum and median of B's ranks is

$$\begin{array}{rcl} sum_B & = & 2.5+4+5.5+8+9.5+11=40.5 \\ median_B & = & 6.75 \end{array}$$

The U statistic is then calculated where $U_x = sum_x - (N_1(N_2+1))/2$:

$$\begin{array}{l} U_A = 25.5 - 5*6/2 = 10.5 \\ U_B = 40.5 - 6*7/2 = 19.5 \end{array}$$

These can be converted to a Z-curve using:

$$\begin{array}{rcll} \mu & = & (N_1N_2)/2 & = & 516.4 \\ \sigma & = & \sqrt{\frac{N_1N_2(N_1+N_2+1)}{12}} & = & 5.477 \\ Z_A & = & (U_A-\mu)/\sigma & = & -0.82 \\ Z_B & = & (U_B-\mu)/\sigma & = & 0.82 \end{array}$$

(Note that Z_A and Z_B have the same absolute value. In all case, these will be the same, with opposite signs.)
If $abs(Z) < 1.96$ then the samples A and B have the same median rankings (at the 95% significance level). In this case, we add one to both $ties_A$ & $ties_B$. Otherwise, their median values can be compared, using some domain knowledge. In this work, *lower* values are better since we are comparing relative errors. Hence:

- If $median_A < median_B$ add 1 to both $wins_A$ & $losses_B$.
- Else if $median_A > median_B$ add 1 to both $losses_A$ & $wins_B$.
- Else, add 1 to both $ties_A$ & $ties_B$.

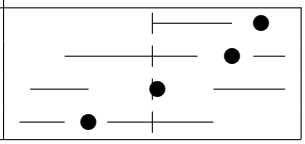
Fig. 6. An example of the Mann-Whitney U test.

were either statistically insignificant (measured using a Mann-Whitney U-test) or the false alarm increased in the intra-experiment.

group	pd(inter)	pf(inter)	data sets	n
a	decreased	decreased	PC5	1
b	same	same	KC3	1
c	same	increased	MC1 PC2	2
d	increased	increased	CM1 KC1 KC2 MC2 MW1 PC1 PC3 PC4	8

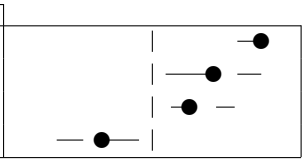
The size of pd increase was quite remarkable. The following chart shows that across our 12 data sets, the top quarter of the pds in the inter-experiemtnt were all in the maximum range of $pd = 100\%$. As predicted by Figure 1, the price of this dramatic increase in pd is an near doubling of the median pf:

	min	q1	median	q3	max
pd / inter	50	80	91	100	100
pd / intra	17	67	80	88	100
pf / inter	4	26	52	73	100
pf / intra	0	17	26	33	73



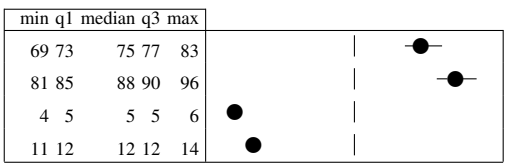
The increase in individual data sets was even larger. For example, in $KC2$, the median pf rose from 73 to 91% while the median pf rose from 31% to 64%:

	min	q1	median	q3	max
pd / inter	82	91	91	100	100
pd / intra	55	73	73	82	91
pf / inter	57	62	64	74	81
pf / intra	14	24	31	33	45



In one data set, a compeltely opposite effect was observed to the main effect. In "PC5", the inter-experiment made the the pd worse and the pf better:

	min	q1	median	q3	max
pd / inter	69	73	75	77	83
pd / intra	81	85	88	90	96
pf / inter	4	5	5	5	6
pf / intra	11	12	12	12	14



V. Incremental Studies

Figure 7 shows the inter-experimental results as the size of the traiing set increased in size. Note two main visual patterns:

- Increasing the training set size does little or nothing to the median $balance$ values.
- In some data sets, increasing the training set size increased the variability (spread) in the $balance$ values.

Mann-Whitney tests were applied to test the first visual pattern. Detectors learned from just N instances do as well as detectors learned from any other number of instances.

- N=100 for {CM1,KC2,KC3,MC2,MW1,PC3,PC4}
- N=200 for {KC1}
- N=300 for {PC1}
- N=4,900 for {PC2} (but N=800 only lost 2 out of 53 trials)
- N=8,300 for {MC1} (but N=400 only lost once out of 92 trials)
- N=11,000 for {PC5} (but N=300 only lost twice out of 169 trials)

VI. Under-sampling Studies

Figure 8 shows the results of an under-sampling stduy where $N \in \{25, 50, 75, ...\}$ defective modules were selected along with an equal N number of defect-free modules. Note the same visual pattern as before: increasing data does not necessarily improve $balance$.

Mann-Whitney tests were applied to test this visual pattern. Detectors learned from just N instances do as well as detectors learned from any other number of instances:

- N=25 for {CM1,KC2,KC3,MC1,MC2,MW1,PC1,PC2}
- N=75 for {PC3} (and N=25 losses three times out of 11).
- N=200 for {PC4} (but N=25 losses once out of 13 trials)
- N=575 for {KC1} (for N=25 only lost once out of 25 trials)
- N=1025 for {PC5} (but at N=25, the ltest lose only once our of 25 trials)

References

[1] T. Menzies, D.Owen, and J. Richardson, "The strangest thing about software," 2007, http://menzies.us/pdf/07strange.pdf.

[2] 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.

[3] M. E, G. Dinakaran, and N. Mosley, "How valuable is it for a web company to use a cross-company cost model, compared to using its own single-company model?" in *16th International World Wide Web Conference, Banff, Canada, May 8-12*, 2007, available from http://www2007.org/paper326.php.

[4] D. Heeger, "Signal detection theory," 1998, available from http://white.stanford.edu/~heeger/sdt/sdt.html.

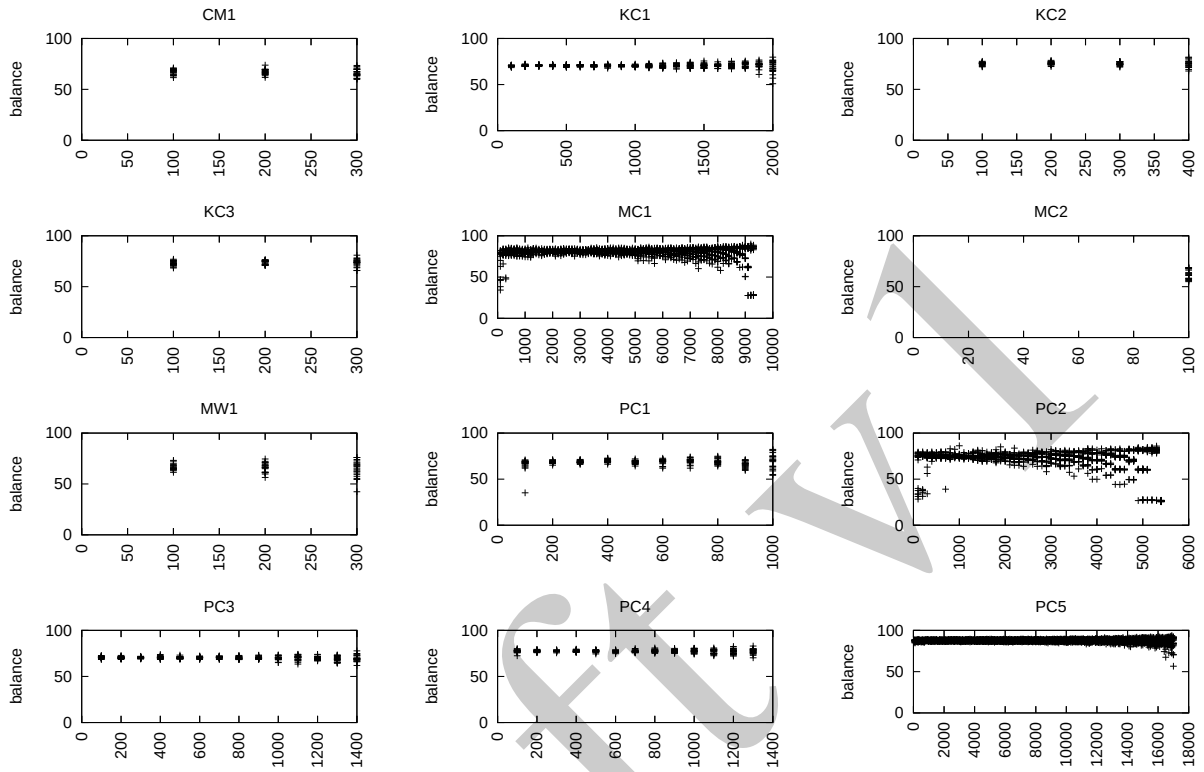


Fig. 7. Incremental Results

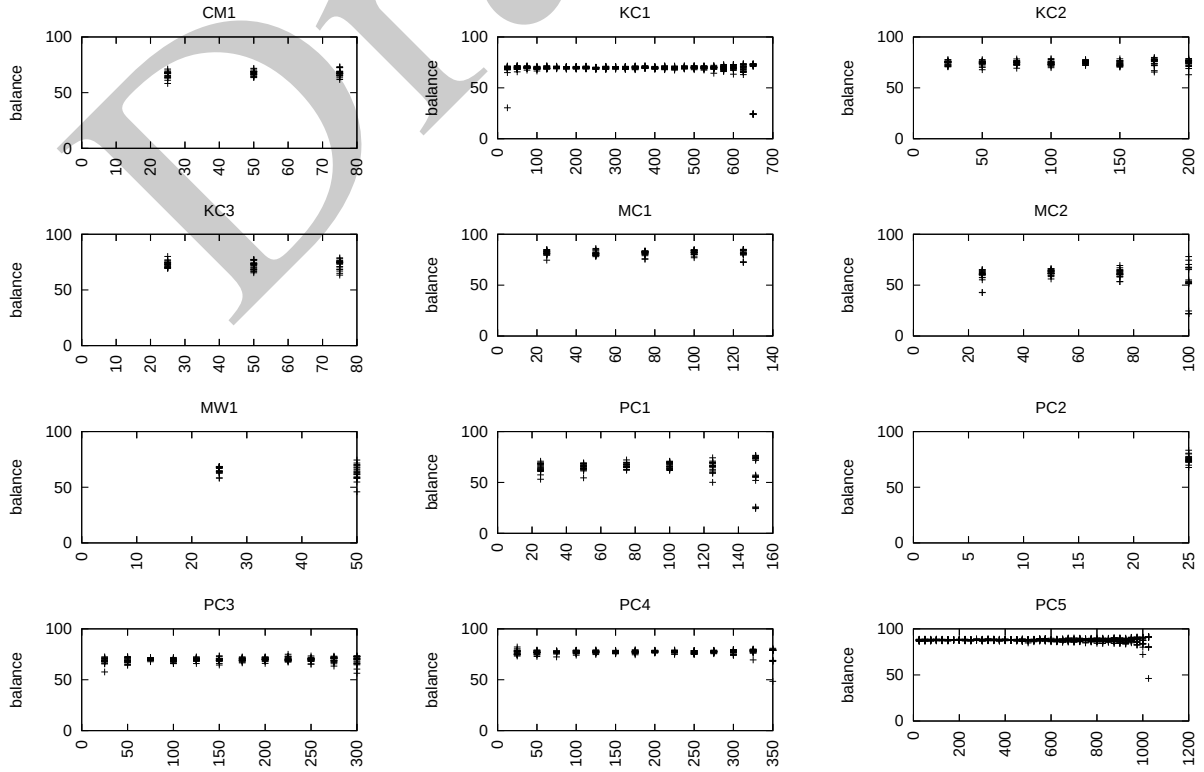


Fig. 8. Under-sampleing results

[5] T. Menzies, J. S. DiStefeno, M. Chapman, and K. Mcgill, "Metrics that matter," in *27th NASA SEL workshop on Software Engineering*, 2002, available from http://menzies.us/pdf/02metrics.pdf.
[6] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, January 2007, available from http://menzies.us/pdf/06learnPredict.pdf.
[7] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Trans. Software Eng*, vol. 32, no. 1, pp. 4–19, 2006. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TSE.2006.3
[8] T. Menzies, J. S. D. Stefano, C. Cunanan, and R. M. Chapman, "Mining repositories to assist in project planning and resource allocation," in *International Workshop on Mining Software Repositories*, 2004, available from http://menzies.us/pdf/04msrdefects.pdf.
[9] T. Menzies and J. S. D. Stefano, "How good is your blind spot sampling policy?" in *2004 IEEE Conference on High Assurance Software Engineering*, 2003, available from http://menzies.us/pdf/03blind.pdf.
[10] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "The detection and classification of non-functional requirements with application to early aspects," in *RE 2006*, 2006, pp. 36–45.
[11] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, October 2002.
[12] G. Antoniol and Y.-G. Gueheneuc, "Feature identification: A novel approach and a case study," in *ICSM 2005*, 2005, pp. 357–366.
[13] A. Marcus and J. Maletic, "Recovering documentation-to-source code traceability links using latent semantic indexing," in *Proceedings of the Twenty-Fifth International Conference on Software Engineering*, 2003.
[14] M. Halstead, *Elements of Software Science*. Elsevier, 1977.
[15] T. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, Dec. 1976.
[16] N. E. Fenton and S. Pfleeger, *Software Metrics: A Rigorous & Practical Approach (second edition)*. International Thompson Press, 1995.
[17] I. H. Witten and E. Frank, *Data mining. 2nd edition*. Los Altos, US: Morgan Kaufmann, 2005.
[18] R. Bouckaert, "Choosing between two learning algorithms based on calibrated tests," in *ICML'03*, 2003, available from http://www.cs.pdx.edu/~timm/dm/10x10way.
[19] J. Demsar, "Statistical comparisons of clasifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006, avaliable from http://jmlr.csail.mit.edu/papers/v7/demsar06a.html.
[20] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics*, vol. 1, pp. 80–83, 1945.
[21] B. Kitchenham, L. Pickard, S. MacDonell, and M. Shepperd, "What accuracy statistics really measure," *Software, IEE Proceedings*, vol. 148, no. 3, pp. 81–85, 2001.
[22] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Ann. Math. Statist.*, vol. 18, no. 1, pp. 50–60, 1947, available online at http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&hand%le=euclid.aoms/1177730491.