When to Use Data from Other Projects for Effort Estimation

Ekrem Kocaguneli, Gregory Gay, Tim Menzies, Ye Yang[1], Jacky Keung[2]
Lane Department of Electrical Engineering and Computer Science, West Virginia University, Morgantown, WV 26506-6109, USA
[1] Institute of Software, Chinese Academy of SciencesBeijing, People's Republic of China
[2] School of Computer Science and Engineering, University of New South Wales, Sydney, Australia

ABSTRACT

Collecting the data required for quality prediction *within* a development team is time-consuming and expensive. An alternative is to make predictions using data that *crosses* from other projects or even other companies. We show that with / without *relevancy filtering*, imported data performs the same / worse (respectively) than using local data. Therefore, we recommend the use of relevancy filtering whenever generating estimates using data from another project.

INTRODUCTION

When data is scarce *within* one project, it is tempting to use data imported from other projects. Such *cross*-project data exist; for example the PROMISE repository [?] offers a dozen effort estimation data sets for public access.
A recent survey paper has evaluated *within* or *cross* data for effort estimation [?]. They concluded that they could not make a conclusion; that the current findings are contradictory about the relative merits of *within* or *cross* data.
In other work [?], we have shown that it is acceptable to use *cross* data sources for defect prediction, providing that data has been pre-processed by some sort of *relevancy filtering*. Given a large training set, such relevancy filters select a small subset relevant to the current test case. Such filtering removes training instances that create noise in the estimation process, leaving a body of data that, in theory, follows the principle of locality.
The success of relevancy filtering for defect prediction prompts us to apply it to effort estimation. To the best of our knowledge, this is the first exploration in the effort estimation community of the effects of relevancy filtering when applied to *cross* and *within* project data. We show that *cross* data can usually attain estimation accuracies just as high as those of *within* data, provided that a relevancy filter is applied to the data, prior to making estimates.

RELEVANCY FILTERING

Our relevancy filter extends standard analogy-based estimation methods (which we call ABE0). ABE0 generates estimates for a test project by gathering evidence from the effort values of similar projects in some training set. By analyzing the previous research of experts like Shepperd et. al.[?], Mendes et. al.[?] and Li et. al.[?] on the field of analogy-based estimation, we can come up with a baseline technique. The baseline technique works as follows:

- Build a training data from rows of past projects;
- The columns of this set are composed of *independent* variables (the features that define projects) and a *dependent* variable (the recorded effort value).
- Decide on how many similar projects (*analogies*) to use when examining a new test instance , i.e. *k*-values.
- For each test instance, select *k* analogies from training.
  - While selecting analogies, use a similarity measure: ABE0 uses the Euclidean distance.

$$Distance = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \qquad (1)$$

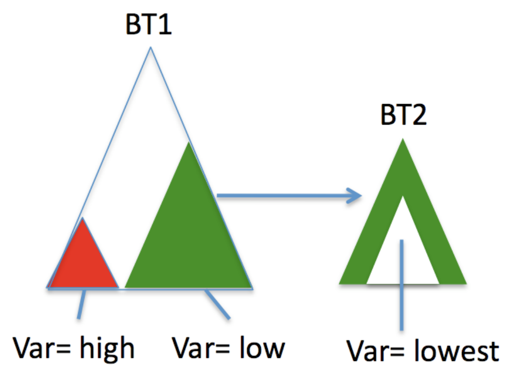
  - Before calculating similarity, apply a scaling measure on features: ABE0 normalizes to 0-1 interval.
- Adapt *k* nearest analogies to calculate an effort estimate: ABE0 takes median of these values.

DETAILS OF RELEVANCY FILTERING

Our relevancy filter is a small variant of ABE0. It is a two-pass system. Pass 1 removes the training instances implicated in poor decisions; pass 2 selects those instances nearest the test instance.

The leaves of the remaining sub-trees are the *survivors* of pass one. These move to pass 2 where the survivors are used to build a second binary tree (called BT2). BT2 is generated and traversed by test instances in the same fashion as BT1. This time, while traversing the tree, instead of storing the variances of sub-trees, we use the variance as a decision criterion. If the variance of the current tree is larger than its sub-trees, then continue to move down the subtree; otherwise, stop moving and select the instances in the current tree as the relevant instances and adapt them for estimation.

This filter is similar to the NN-filter used by Turhan et.al.[?], except that there is no need to pre-specify the number of analogies k to be used for estimation. Each test instance selects its own relevant analogies by traversing to different sub-trees of BT2. For a detailed discussion on the rationale behind this filter, see [?]. All we need to say here is that this filter is known to generate low errors for ABE0-style effort estimation [?]. Hence, it is a suitable tool for the rest of this study.



Each tree BT1 and BT2 are binary cluster trees. The red sub-tree is pruned in pass one due to high variance. The remaining subtrees (shown in green) form the right-hand tree.
In pass two, test instances start at the root of this tree and traverse to the nearest child (and so on, recursively). While the sub-tree variance continues to decrease, the traversal continues. Estimates are generated from the median of the instances of the right-hand-side sub-tree with lowest variance.

METHODOLOGY

In our research, we have used subsets of three commonly-used datasets in software effort estimation research: Nasa93, the original Cocomo81 [?], and Desharnais[?]. We will denote the subsets of

Nasa93 as Nasa93c1, Nasa93c2 and Nasa93c5 that contain projects from development centers 1, 2 and 5 respectively. In a similar fashion, subsets of Cocomo81 will be denoted as Coc81o, Coc81e and Coc81s (for organic, embedded, and semidetached). Finally, the Desharnais dataset is split into three different subsets: DesL1, DesL2 and DesL3 (languages 1, 2 and 3 respectively). Since each of these subsets have certain common criteria (the development center, development mode, or development language), each subset will be treated as a separate *within* dataset. All of the datasets used in this research are available in PROMISE data repository [?].

For each of the three main datasets (Nasa93, Cocomo81 and Desharnais) in our research, we have conducted *within* and *cross* experiments. Each subset became a *within* dataset that contains projects sharing the particular characteristics of a single development firm.

To understand the *within* and *cross* data formation, assume that a dataset X with its three subsets X_1, X_2 and X_3 is under consideration. For *within* experiments, relevancy filtering described is applied on each one of X_1, X_2 and X_3 separately and the median of the filtered project instances in the training set is stored as the effort estimate for the test instance. For the *cross* experiments, one of X_1, X_2 or X_3 is chosen as the test set and the combination of the remaining two forms the *cross* dataset for training. This time, the relevancy filtering is applied on the *cross* dataset, and the estimations for projects in the test set are stored. Each of the *within* and *cross* experiments are repeated twenty times in order to remove any bias that would otherwise be brought by a particular test and training set combination.

In order to compare the performance of *within* and *cross* datasets, we have used two measures: the magnitude of relative error (MRE) and win-tie-loss values generated by a statistical rank-sum test. Using a Mann-Whitney test (95%), we checked how often one treatment won / lost / tied with the others. Here, a "tie" means that they are not statistically significant different. If statistically different, then the method with a lower median MRE score gets one more "win" and the other method gets one more "loss".

RESULTS-Without Relevancy Filtering

Data set	Train Set	Test Set	Method	Win	Tie	Loss
Nasa93	Nasa93c1	Leave-one-out test instance	*Within*	1		
	Nasa93c2 and Nasa93c5	Nasa93c1	*Cross*			1
	Nasa93c2	Leave-one-out test instance	*Within*	1		
	Nasa93c1 and Nasa93c5	Nasa93c2	*Cross*			1
	Nasa93c5	Leave-one-out test instance	*Within*		1	
	Nasa93c1 and Nasa93c2	Nasa93c5	*Cross*		1	
Cocomo81	Coc81o	Leave-one-out test instance	*Within*	1		
	Coc81e and Coc81s	Coc81o	*Cross*			1
	Coc81e	Leave-one-out test instance	*Within*		1	
	Coc81o and Coc81s	Coc81e	*Cross*		1	
	Coc81s	Leave-one-out test instance	*Within*		1	
	Coc81o and Coc81e	Coc81s	*Cross*		1	
Desharnais	DesL1	Leave-one-out test instance	*Within*	1		
	DesL2 and DesL3	DesL1	*Cross*			1
	DesL2	Leave-one-out test instance	*Within*		1	
	DesL1 and DesL3	DesL2	*Cross*		1	
	DesL3	Leave-one-out test instance	*Within*	1		
	DesL1 and DesL2	DesL3	*Cross*			1

Every odd and even line is a pair of experiments. In each pair, there is a *within* and a *cross* experiment. In *cross* experiment, a linear regression model is built on *cross* data and tested on the *within* data. In *within* experiment, the test instance is selected with leave-one-out, and a linear regression model is built on the remaining instances and tested on the selected test instance. A "1" denotes which item in the pair won, lost or tied.

RESULTS-With Relevancy Filtering

Nasa93 Results

Dataset	Method	Win	Tie	Loss
Nasa93c1	within	3	15	2
Nasa93c2 and Nasa93c5	cross	2	15	3
Nasa93c2	within	3	17	0
Nasa93c1 and Nasa93c5	cross	0	17	3
Nasa93c5	within	1	19	0
Nasa93c1 and Nasa93c2	cross	0	19	1

Figure above, shows the the win-tie-loss values for the subsets of Nasa93. This result shows us that, in all three treatments, the *tie* values are quite high. This indicates that, for at least 75% of the tests, there is no statistical difference between filtered *cross* and *within* results.

Cocomo81 Results

Dataset	Method	Win	Tie	Loss
Coc81o	within	13	7	0
Coc81e and Coc81s	cross	0	7	13
Coc81e	within	1	19	0
Coc81o and Coc81s	cross	0	19	1
Coc81s	within	0	20	0
Coc81o and Coc81e	cross	0	20	0

Above Figure shows the win-tie-loss values for the subsets of Cocomo81. In two out of the three treatments the *tie* values are 19, which tells us that for these treatments, *within* and *cross* performance are almost identical. However, the first treatment shows a preference for *within* data on thirteen of the twenty tests.

Desharnais Results

Dataset	Method	Win	Tie	Loss
DesL1	within	1	19	0
DesL2 and DesL3	cross	0	19	1
DesL2	within	1	19	0
DesL1 and DesL3	cross	0	19	1
DesL3	within	16	4	0
DesL1 and DesL2	cross	0	4	16

The win-tie-loss values for subsets of Desharnais are given in the Figure above. Two out of the three treatments show identical *tie* values of 19, which again suggests that the performance of filtered *cross* datasets is statistically identical to *within* datasets.

In the majority case ($\frac{7}{9}$ treatments) the *cross* data performs as well as the *within* data for effort estimation. There are only two treatments, DesL3 and Coc81o, where *within* performance was significantly better than *cross* performance, whose explanation can be hidden in the dataset size, but with currently-available information it is difficult to suggest any conclusive reason for the situation.

SUPPORT

This research was carried out at West Virginia University under a grant from the National Science Foundation (NSF CISE 71608561).