When to Use Data from Other Projects for Effort Estimation

Ekrem Kocaguneli, Gregory Gay, Tim Menzies
LCSEE, WVU, USA
ekocagun@mix.wvu.edu, greg@greggay.com, tim@menzies.us

Ye Yang
Institute of Software
Chinese Academy of Sciences
Beijing, People's Republic of China
ye@itechs.iscas.ac.cn

Jacky W. Keung
School of Computer Science and Engineering
University of New South Wales
Sydney, Australia
jacky.keung@nicta.com.au

ABSTRACT

Collecting the data required for quality prediction *within* a development team is time-consuming and expensive. An alternative 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.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Effort Estimation

Keywords

Effort estimation, data mining, cross, within

1. 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 [2] offers a dozen effort estimation data sets for public access.

A recent survey paper has evaluated *within* or *cross* data for effort estimation [5]. 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 [10], 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.

ASE '10 Antwerp, Belgium

2. 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. [9], Mendes et. al. [8] and Li et. al. [7] on the field of analogy-based estimation, we can come up with a baseline technique:

- 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 (such as the Euclidean distance of features).
  - Before calculating similarity, apply a scaling measure on independent features to equalize their influence on this similarity measure.
- Use the effort values of the *k* nearest analogies to calculate an effort estimate.

We can refer to this baseline framework as ABE0. For the similarity measure, we use Euclidean distance:

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

ABE0 returns the median of the efforts in the *k* nearest analogies.

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.

In pass 1, the training projects are used to generate a binary tree. The leaves of this binary tree are formed by the individual training projects, which are then greedily clustered in tuples to form the parent levels. This binary tree, which we will call BT1, is then traversed upwards from the root to height level one (one higher level than the leaves). The variance of the effort values in each subtree (the performance variance) is then recorded and normalized to a 0-1 interval. Pass one prunes all sub-trees with a variance greater than 10% of the maximum variance seen in any tree.

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

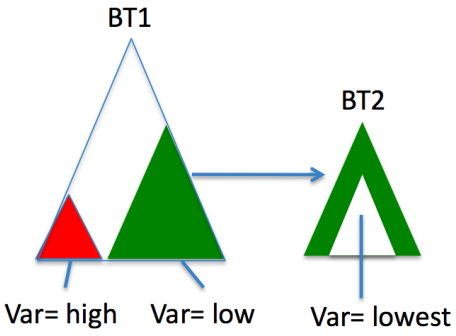

Figure 1: Two pass relevancy filtering. Each tree BT1 and BT2 are binary cluster trees. The red sub-tree is pruned in pass one due to high variance. The remaining subtrees (shown in green) form the right-hand tree. In pass two, test instances start at the root of this tree and traverse to the nearest 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.

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. Relevancy filtering is visualized in Figure 1.

This filter is similar to the NN-filter used by Turhan et.al. [10], 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 [6]. All we need to say here is that this filter is known to generate low errors for ABE0-style effort estimation [6]. Hence, it is a suitable tool for the rest of this study.

3. METHODOLOGY

In our research, we have used subsets of three commonly-used datasets in software effort estimation research: Nasa93, the original Cocomo81 [1], and Desharnais [4].

We will denote the subsets of Nasa93 as Nasa93c1, Nasa93c2 and Nasa93c5. Nasa93c1, Nasa93c2 and Nasa93c5 contain projects from different NASA development centers around the United States (denoted as development centers 1, 2 and 5 in the complete dataset). 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 [2].

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, the relevancy filtering described in Section 2 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 separation of training and testing sets, the leave-one-out method is used. Leave-one-out selects one instance out of a dataset of n instances as the test instance and uses the remaining $n-1$ instances as the training set.

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.

$$MRE = \frac{|actual_i - predicted_i|}{actual_i} \quad (2)$$

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".

4. RESULTS

In our experiment, we analyzed *3 datasets * 3 subsets = 9 treatments*. We evaluated *cross* and *within* performances of each particular dataset, subject to statistical tests, with and without relevancy filtering.

4.1 Without Relevancy Filtering

In this first experiment, we have 9 treatments and for each treatment we observe the estimation performances when *within* and *cross* datasets are used. For this purpose we used a linear regression model. Two-pass filtering was not applied.

In *cross* experiments, for each data set, we selected one of the 3 subsets as the test set and the remaining two as the train set. We then built a linear regression model on the *cross* data and applied this model on the test set. For the *within* dataset we also used a linear regression model. The test case selection for *within* experiment is performed in accordance with leave-one-out method, which picks up one of the instances in the dataset as the test set and uses the remaining instances as the train set. The linear regression model that is built on the train set is then tested on the single test instance. After applying linear regression model on *within* and *cross* datasets, we calculated the win-tie-loss values for each treatment.

As shown in Figure 2, we see that in a minority of cases ($\frac{4}{9}$, see Nasa93c5, Coc81e, Coc81s and DesL2), *cross* and *within* data perform just as well as each other. In the majority case ($\frac{5}{9}$, see Nasa93c1, Nasa93c2, Coc81o, DesL1, DesL3), *within* performed better than *cross* data. That is, in the absence of relevancy filtering, the *within* datasets yield significantly lower MRE values in majority of cases.

4.2 With Relevancy Filtering

This section shows that for each data set, the application of relevancy filtering reverses the conclusion of the previous section; i.e.

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

Figure 2: MRE win-tie-loss results without relevancy filtering. 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.

the *cross* data becomes useful for estimating the local project.

Figure 3, shows the the win-tie-loss values for the subsets of Nasa93. The greedy clustering algorithm of the two pass relevancy filtering uses some non-determinism (when breaking ties between instances of similar distances), so we repeat these experiments twenty times.

This results 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. In short, for Nasa93, the performance of *cross* data (filtered for relevancy) is indistinguishable from the performance of *within* data.

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 3: MRE win-tie-loss values for Nasa93 from 20 *randomized assessments*. In all treatments tie values are quite high. For Nasa93, the performance of *cross* data is mostly same as *within* data.

Figure 4 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.

The win-tie-loss values for subsets of Desharnais are given in Figure 5. The derived results for the Desharnais subsets are similar to those of Cocomo81 treatments: 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. However, in one of the treatments, *within* outperforms *cross* on sixteen of the twenty trials.

In summary, with relevancy filtering, 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. A possible explanation for those two scenarios may be hidden in the dataset size or in the quality of the *within* datasets, but the currently-available information makes it difficult to suggest any conclusive reason for the situation.

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

Figure 4: MRE win-tie-loss values for Cocomo81 from 20 *randomized assessments*. In 2 treatments *cross* data is the same as the *within* data. However, in the case of Coc81o, *within* outperforms *cross* data.

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

Figure 5: MRE win-tie-loss values for Desharnais from 20 *randomized assessments*. In the case of DesL3 the *within* data is much better than the *cross* data. For other treatments, *within* and *cross* data are statistically the same.

Cross Dataset	Test Set	Instances selected in BT2		
		From Nasa93c1	From Nasa93c2	From Nasa93c5
Nasa93c2 and Nasa93c5	Nasa93c1	0	2.4	1.2
Nasa93c1 and Nasa93c5	Nasa93c2	1.4	0	1.8
Nasa93c1 and Nasa93c2	Nasa93c5	2.4	1.1	0
		From Coc81o	From Coc81e	From Coc81s
Coc81e and Coc81s	Coc81o	0	2.0	1.3
Coc81o and Coc81s	Coc81e	3.6	0	1.4
Coc81o and Coc81e	Coc81s	2.8	0.3	0
		From DesL1	From DesL2	From DesL3
DesL2 and DesL3	DesL1	0	2.3	0.7
DesL1 and DesL3	DesL2	2.1	0	0.1
DesL1 and DesL2	DesL3	3.2	1.6	0

Figure 6: Mean number of instances used for estimation after filtering in 20 runs. *Cross* datasets are combinations of two *within* datasets tested on another *within* dataset.

5. DISCUSSION

Figure 6 shows the mean number of instances used for analogy-based estimation by our two-pass relevancy filtering algorithm. Surprisingly, the number of selected analogies is very small: mean value around 3. Further, while exceptions exist, the selected analogies come from multiple other projects. For example, for the Nasa93 dataset, the data relevant to center c1 came $\frac{2}{3}$-rds and $\frac{1}{3}$-rd from centers c2 and c5 (respectively).

This suggests that we should revisit what we mean by *within* and *cross*:

- Effort estimation functions on a theory of *locality*; i.e. new projects follow similar practices to historical projects and should require a similar amount of effort. As Chen et. al. [3] have shown, inconsistencies in data collection across multiple companies create locality-specific biases in *cross* data sets. Such biases result in an unacceptable amount of variance in the effort calculations.
- Figure 6 is saying that *similar* projects may not come from the same geographical location. Our relevance filters hunted out handfuls of *similar* projects from other subsets. Perhaps software differs on many dimensions (such as where it was written), and the most important dimensions for finding *similar* projects may not be mere geography.

Given the complex multi-dimensional nature of the software creation process, the geographical dimension may be less important than other factors. The most similar software to what you are writing now may not be in the next office. Rather, it may be in an office on the other side of the world.

Going forward, we would like to learn exactly why an instance is deemed "relevant" or "irrelevant" by our filter. In other words, we would like to know exactly which features are most influential when assigning relevancy. The ability to identify these exact dimensions would make the selection of appropriate projects easier for any institution that uses *cross* data. It would also lead to (a) more accurate filtering techniques; and (b) a better understanding of the structure of software projects including where to find data most relevant to some current project.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China under Grant Nos. 60873072 and 90718042.

6. REFERENCES

[1] B. W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[2] G. Boetticher, T. Menzies, and T. Ostrand. PROMISE repository of empirical software engineering data, 2007.

[3] J. Chen, Y. Yang, V. Nguyen, and Q. Li. Reducing the local bias in calibrating the general cocomo. *International Forum on COCOMO and Systems-Software Cost Modeling*, 2009.

[4] J. Desharnais. Analyse statistique de la productivitie des projets informatique a partie de la technique des point des fonction. Master's thesis, Univ. of Montreal, 1989.

[5] B. A. Kitchenham, E. Mendes, and G. H. Travassos. Cross versus within-company cost estimation studies: A systematic review. *IEEE Trans. Softw. Eng.*, 33(5):316–329, 2007.

[6] E. Kocaguneli. Better methods for configuring case-based reasoning systems. Master's thesis, 2010.

[7] Y. Li, M. Xie, and T. Goh. A study of project selection and feature weighting for analogy based software cost estimation. *Journal of Systems and Software*, 82:241–252, 2009.

[8] E. Mendes, I. D. Watson, C. Triggs, N. Mosley, and S. Counsell. A comparative study of cost estimation models for web hypermedia applications. *Empirical Software Engineering*, 8(2):163–196, 2003.

[9] M. Shepperd, C. Schofield, and B. Kitchenham. Effort estimation using analogy. In *International Conference on Software Engineering*, pages 170–178, 1996.

[10] B. Turhan, T. Menzies, A. Bener, and J. Di Stefano. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, 14:540 – 578, 2009.