A Ranking Stability Indicator for Selecting the Best Effort Estimator in Software Cost Estimation

Jacky Keung · Ekrem Kocaguneli · Tim Menzies

Received: 24 December 2010 / Accepted: February 2011
Springer Science+Business Media, LLC 2011

Abstract Software effort estimation research shows that there is no universal agreement on the “best” effort estimation approach. This is largely due to the “ranking instability” problem, which is highly contingent on the evaluation criteria and the subset of the data used in the investigation. There are a large number of different method combination exists for software effort estimation, selecting the most suitable combination becomes the subject of research in this paper. Unless we can reasonably determine stable rankings of different estimators, we cannot determine the most suitable estimator for effort estimation. This paper reports an empirical study using 90 estimation methods applied to 20 datasets as an attempt to address this question. Performance was assessed using MAR, MMRE, MMER, MBRE, MIBRE, MdMRE, PRED(25) and compared using a Wilcoxon ranked test (95%). An comprehensive empirical experiment was carried out. Result shows prior studies of ranking instability of effort estimation approaches may have been overly pessimistic. Given the large number of datasets, it is now possible to draw stable conclusions about the relative performance of different effort estimation methods and to select the most suitable ones for the study under investigation. In this study, regression trees or analogy-based methods are the best performers in the experiment, and we recommend against neural nets or simple linear regression. Based on the proposed evaluation method, we are able to determine the most suitable local estimator for software cost estimation, an important process in the application of any effort estimation analysis.

Keywords Effort estimation, Data mining, Stability, Linear Regression, Regression Trees, Neural Nets, Analogy, MMRE, Evaluation Criteria

Jacky Keung
Department of Computing
The Hong Kong Polytechnic University
Kowloon, Hong Kong
E-mail: Jacky.Keung@comp.polyu.edu.hk

Ekrem Kocaguneli and Tim Menzies
Lane Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV 26505, USA
E-mail: ekocagun@mix.wvu.edu, tim@menzies.us
1 Introduction

Being able to choose the most suitable software development effort estimator for the local software projects remains illusive for many project managers. For decades, researchers have been searching for the “best” software development effort estimator. At the time of writing, no such “best” estimator has been found which provides consistently the most accurate estimate. The usual conclusion is that effort estimation suffers from a ranking instability syndrome; i.e. different researchers offer conflicting rankings as to what is “best” [30, 32]. It seems, different set of best effort estimators exist under various different situations given different historical sample datasets.

This is an open and urgent issue since accurate effort estimation is vital to Software Engineering, and is often a challenging task for many software project managers. Both over-estimating and underestimating would result unfavorable impacts to the business’s competitiveness and project resource planning. Conventionally, the single most familiar effort estimator may be used for different situations, this approach may not produce the best effort estimates for different projects.

Being able to compare and determine the best effort estimator for different scenarios is critically important to the relevance of the estimates to the target problem under investigation. Software effort estimation research focuses on the learner used to generate the estimate (e.g. linear regression, neural nets, etc) in many cases, overlooking the importance of the quality and characteristics of the data being used in the estimation process. We argue that this approach is somewhat misguided since, as shown below, learner performance is greatly influenced by the data preprocessing and the datasets being used to evaluate the learner.

Ranking stability in software effort estimation should be the primary research focus, being able to correctly classify the characteristics of each method allows the most suitable estimators to be used in the estimation process. This paper presents a method which can be used to determine the best effort estimators to use at different situations.

Method combinations can produce vast different results, in all, this study applies 90 estimators (10 learners and 9 preprocessors) to 20 datasets and measure their performance using seven performance criteria. To the best of our knowledge, this is the largest effort estimation study yet reported in the literature. One result of exploring such a large space of data and algorithms is that we are able to report stable conclusions (while prior studies have not).

This paper is structured as follows. Section 2 addresses our research challenge and motivation. Related work discusses effort estimation and the prior reports on conclusion instability. Those reports used a dataset to seed the generation of artificial data. Our results section shows that if we extend the experiments to a broader set of methods and project data, we are able to discover stable conclusions such as that we can list best (and worst) effort estimators.

2 Searching for the Best Estimator

A result of a classification/ranking procedure is a list of performance indicators, ranked according to their relevance to the target problem. Unlike dataset feature subset selection, there is no consolidated theory exists in literature for estimator selection stability. Ranked estimator lists are highly unstable in the sense that different method combining with different preprocessors may yield very different rankings, and that a small change of the data set usually affects the obtained estimator list considerably. The estimator ranking stability issue
has not been considered for its importance in the literature, but unfortunately, the issue has not grown into the main focus of research in the last few years, perhaps as a consequence of immediate benefits of individual development of estimators claimed to be more superior than the others, but limited to a very specific circumstance.

Without being able to understand the ranking stability, it is unlikely to progress the research in the area of software cost estimation, as a consequence there is not convincing evidence to support the practical usage of the developed methods and tools available in the literature.

To derive stable rankings about which estimator is “best”, there have been attempts in trying to compare model prediction performance of different approaches. For example, Shepperd and Kododa [32] compared regression, rule induction, nearest neighbor and neural nets, in an attempt to explore the relationship between accuracy, choice of prediction system, and different dataset characteristics by using a simulation study based on artificial datasets. They also reported a number of conflicting results exist in the literature as to which method provides superior prediction accuracy, and offered possible explanations including the use of an evaluation criteria such as MMRE or the underlying characteristics of the dataset being used can have a strong influence upon the relative effectiveness of different prediction models. Their work as a simulation study that took a single dataset, then generated very large artificial datasets using the distributions seen on that data. They concluded that:

- None of these existing estimators were consistently “best”;
- The accuracy of an estimate depends on the dataset characteristic and a suitable prediction model for the dataset.

They conclude that it is generally infeasible to determine which prediction technique is the “best”.

Recent results suggest that it is appropriate to revisit the ranking instability hypothesis. Menzies et al. [28] applied 158 estimators to various subsets of two COCOMO datasets. In a result consistent with Shepperd and Kododa, they found the precise ranking of the 158 estimators changed according to the random number seeds used to generate train/test sets; the performance evaluation criteria used; and which subset of the data was used. However, they also found that four methods consistently outperformed the other 154 across all datasets, across 5 different random number seeds, and across three different evaluation criteria.

There are now many datasets in public domain readily available for stability studies. Figure 1 lists 20 datasets which have become available in the last year at the PROMISE repository of reusable SE data1. It is no longer necessary to work on simulated data (as done by Shepperd and Kododa [32]) or to study merely two datasets (as done by Menzies et al. [28]).

When previous studies and conclusions are considered, unless we address the instability issue, we cannot make conclusive remarks about neither the algorithms nor the datasets. Our fundamental motivations is to question the stability issue and we propose a methodology for evaluating the stability (see methodology of Figure 6). Given that methodology, we will propose that if the mean-rank change among methods or datasets is $x$, then we will need a dataset or algorithm amount of more than 2 times the value of this $x$ to be able to make stable conclusions, which allows us to select the correct estimator for software cost estimation.

---

1 http://promisedata.org/data
### Historical Effort Data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Features</th>
<th>Size</th>
<th>Description</th>
<th>Units</th>
<th>Min</th>
<th>Median</th>
<th>Mean</th>
<th>Max</th>
<th>Skewness</th>
</tr>
</thead>
<tbody>
<tr>
<td>cocomo3y</td>
<td>17</td>
<td>63</td>
<td>NASA projects</td>
<td>months</td>
<td>6</td>
<td>98</td>
<td>683</td>
<td>11400</td>
<td>4.4</td>
</tr>
<tr>
<td>cocomo3ye</td>
<td>17</td>
<td>28</td>
<td>Cocomo3y embedded projects</td>
<td>months</td>
<td>9</td>
<td>354</td>
<td>1153</td>
<td>11400</td>
<td>3.4</td>
</tr>
<tr>
<td>cocomo3yo</td>
<td>17</td>
<td>24</td>
<td>Cocomo3y organic projects</td>
<td>months</td>
<td>6</td>
<td>46</td>
<td>60</td>
<td>240</td>
<td>1.7</td>
</tr>
<tr>
<td>nasa93</td>
<td>17</td>
<td>93</td>
<td>NASA projects</td>
<td>months</td>
<td>5.9</td>
<td>156</td>
<td>849.65</td>
<td>6000</td>
<td>2.64</td>
</tr>
<tr>
<td>nasa93_center_1</td>
<td>17</td>
<td>12</td>
<td>NASA93 projects from center 1</td>
<td>months</td>
<td>8</td>
<td>252</td>
<td>624</td>
<td>8211</td>
<td>4.2</td>
</tr>
<tr>
<td>nasa93_center_2</td>
<td>17</td>
<td>37</td>
<td>NASA93 projects from center 2</td>
<td>months</td>
<td>24</td>
<td>66</td>
<td>139.92</td>
<td>360</td>
<td>0.86</td>
</tr>
<tr>
<td>nasa93_center_5</td>
<td>17</td>
<td>40</td>
<td>NASA93 projects from center 5</td>
<td>months</td>
<td>8</td>
<td>82</td>
<td>223</td>
<td>1150</td>
<td>2.4</td>
</tr>
<tr>
<td>desharnais</td>
<td>12</td>
<td>81</td>
<td>Canadian software projects</td>
<td>hours</td>
<td>546</td>
<td>364.7</td>
<td>9046</td>
<td>23940</td>
<td>2.0</td>
</tr>
<tr>
<td>desharnais1</td>
<td>11</td>
<td>46</td>
<td>Projects in Desharnais that are developed with Language1</td>
<td>hours</td>
<td>805</td>
<td>4035.5</td>
<td>5738.9</td>
<td>23940</td>
<td>2.09</td>
</tr>
<tr>
<td>desharnais2</td>
<td>11</td>
<td>25</td>
<td>Projects in Desharnais that are developed with Language2</td>
<td>hours</td>
<td>1155</td>
<td>3472</td>
<td>5116.7</td>
<td>14973</td>
<td>1.16</td>
</tr>
<tr>
<td>desharnais3</td>
<td>11</td>
<td>10</td>
<td>Projects in Desharnais that are developed with Language3</td>
<td>hours</td>
<td>546</td>
<td>1123.5</td>
<td>1684.5</td>
<td>5880</td>
<td>1.86</td>
</tr>
<tr>
<td>sdr</td>
<td>22</td>
<td>24</td>
<td>Turkish software projects</td>
<td>months</td>
<td>2</td>
<td>12</td>
<td>32</td>
<td>342</td>
<td>3.9</td>
</tr>
<tr>
<td>albrecht</td>
<td>7</td>
<td>24</td>
<td>Projects from IBM</td>
<td>months</td>
<td>1</td>
<td>12</td>
<td>22</td>
<td>105</td>
<td>2.2</td>
</tr>
<tr>
<td>finnish</td>
<td>8</td>
<td>38</td>
<td>Software projects developed in Finland</td>
<td>hours</td>
<td>460</td>
<td>5430</td>
<td>7678.3</td>
<td>24670</td>
<td>0.95</td>
</tr>
<tr>
<td>kemerer</td>
<td>7</td>
<td>15</td>
<td>Large business applications</td>
<td>months</td>
<td>23.2</td>
<td>130.3</td>
<td>2192.4</td>
<td>11073</td>
<td>2.76</td>
</tr>
<tr>
<td>maxwell</td>
<td>27</td>
<td>62</td>
<td>Projects from commercial banks in Finland</td>
<td>hours</td>
<td>583</td>
<td>5189.5</td>
<td>8223.2</td>
<td>63694</td>
<td>3.26</td>
</tr>
<tr>
<td>miyazaki94</td>
<td>8</td>
<td>48</td>
<td>Japanese software projects developed in COBOL</td>
<td>months</td>
<td>5.6</td>
<td>38.1</td>
<td>87.47</td>
<td>1586</td>
<td>6.06</td>
</tr>
<tr>
<td>telecom</td>
<td>3</td>
<td>18</td>
<td>Maintenance projects for telecom companies</td>
<td>months</td>
<td>23.54</td>
<td>222.53</td>
<td>284.33</td>
<td>11155</td>
<td>1.78</td>
</tr>
<tr>
<td>china</td>
<td>18</td>
<td>599</td>
<td>Projects from Chinese software companies</td>
<td>hours</td>
<td>26</td>
<td>1829</td>
<td>3921</td>
<td>54620</td>
<td>3.92</td>
</tr>
</tbody>
</table>

Total: 1198
3 Estimation Methods for Software Development Projects

This section reviews the effort estimation literature with regards to (a) the major estimation techniques used by empirical research studies on cost estimation within the last 15 years and (b) the conclusion instability problem.

3.1 Algorithmic Methods

There are many algorithmic effort estimators. For example, if we restrict ourselves to just instance-based algorithms, Figure 2 shows that there are thousands of options just in that one sub-field.

As to non-instance methods, there are many proposed in the literature including various kinds of regression (simple, partial least square, stepwise, regression trees), and neural networks just to name a few. For notes on these non-instance methods, see §4.3.

Note that instance & non-instance-based methods can be combined to create even more algorithms. For example, once an instance-based method finds its nearest neighbors, those neighbors might be summarized with regression or neural nets [25].

3.2 Non-Algorithmic Methods

An alternative approach to algorithmic approaches (e.g. the instance-based methods of Figure 2) is to utilize the best knowledge of an experienced expert. Expert based estimation [13] is a human intensive approach that is most commonly adopted in practice. Estimates are usually produced by a domain expert based on their very own personal experience. It is flexible and intuitive in a sense that it can be applied in a variety of circumstances where other estimating techniques do not work (for example when there is a lack of historical data). Furthermore in many cases requirements are simply unavailable at the bidding stage of a project where a rough estimate is required in a very short period of time.

Jorgensen [14] provides guidelines for producing realistic software development effort estimates derived from industrial experience and empirical studies. One important finding concluded was that the combined estimation method in expert based estimation offers the most robust and accurate combination method, as combining estimates captures a broader range of information that is relevant to the target problem, for example combining estimates of analogy based with expert based method. Data and knowledge relevance to the project’s context and characteristics are more likely to influence the prediction accuracy.

Although widely used in industry, there are no standard methods for expert based estimation. Shepperd et al. [34] do not consider expert based estimation an empirical method because the means of deriving an estimate are not explicit and therefore not repeatable, nor easily transferable to other staff. In addition, knowledge relevancy is also a problem, as an expert may not be able to justify estimates for a new application domain. Hence, the rest of this paper does not consider non-algorithmic methods.

4 Experiment Design

In our experiments, numerous performance measures were collected after various algorithms (combinations of preprocessors and learners) were applied to the data of Figure 1. This section describes those performance measures, preprocessors, and learners.
Instances-based learners draw conclusions from instances near the test instance. Mendes et al. [27] discuss various near-ness measures.

- $M_1$: A simple Euclidean measure;
- $M_2$: A “maximum distance” measure that focuses on the single feature that maximizes inter-project distance;
- $M_3$: More elaborate kernel estimation methods.

Once the nearest neighbors are found, they must be used to generate an effort estimate via...

- $R_1$: Reporting the median effort value of the analogies;
- $R_2$: Reporting the mean dependent value;
- $R_3$: Reporting a weighted mean where the nearer analogies are weighted higher than those further away [27];

Prior to running an instance-based learning, it is sometimes recommended to handle anomalous rows by:

- $N_1$: Doing nothing at all;
- $N_2$: Using outlier removal [18];
- $N_3$: Prototype generation; i.e. replace the data set with a smaller set of most representative examples [8].

When computing distances between pairs, some feature weighting scheme is often applied:

- $W_1$: All features have uniform weights;
- $W_2$-$W_9$: Some pre-processing scores the relative value of the features using various methods [12,18,25].

The pre-processors may require discretization. Discretization breaks up continuous ranges at points $b_1, b_2, ..., b_k$, each containing counts of $c_1, c_2, ...$ of numbers [11]. Discretization methods include:

- $D_1$: Equal-frequency, where $c_i = c_j$;
- $D_2$: Equal-width, where $b_{i+1} - b_i$ is a constant;
- $D_3$: Entropy [9];
- $D_4$: PKID [36];
- $D_5$: Do nothing at all.

Finally, there is the issue of how many $k$ neighbors should be used:

- $K_1$: $k = 1$ is used by Lipowezky et al. [26] and Walkerden & Jeffery [35];
- $K_2$: $k = 2$ is used by Kirropp & Shepperd [19];
- $K_3$: $k = 1, 2, 3$ is used by Mendes et al. [27];
- $K_4$: Li et al. use $k = 5$ [25];
- $K_5$: Baker tuned $k$ to a particular training set using an experimental method [3].

Fig. 2: Each combination of the above $N \times W \times D \times M \times R \times K$ techniques is one algorithm for instance-based effort estimation. This figure shows $3 \times 3 \times 3 \times 9 \times 5 \times 5 > 6,000$ algorithms for effort estimation. Some of these ways can be ruled out, straight away. For example, at $k = 1$, then all the adaptation mechanisms return the same result. Also, not all the feature weighting techniques require discretization, decreasing the space of options by a factor of five. However, even after discarding some combinations, there are still hundreds to thousands of algorithms to explore.

Since it is impractical to explore (say) the thousands of options described in Figure 2, we elected to explore variants commonly used in the literature. For example, we explore neural nets, regression, and analogy because those methods were explored by Shepherd and Kododa [32]. Nevertheless, it is important to note that our conclusions come only from the estimators/performance criteria/datasets used in this study. Further work is required to confirm our findings on other estimators/performance criteria/datasets.
4.1 Performance Measures

Performance measures comment on the success of a prediction. For example, the absolute residual (AR) is the difference between the predicted and the actual:

$$AR_i = x_i - \hat{x}_i$$

(1)

(where \(x_i, \hat{x}_i\) are the actual and predicted value for test instance \(i\)).

The Magnitude of Relative Error measure a.k.a. MRE is a very widely used evaluation criterion for selecting the best effort estimator from a number of competing software prediction models [33] [10]. MRE measures the error ratio between the actual effort and the predicted effort and can be expressed as the following equation:

$$MRE_i = \frac{|x_i - \hat{x}_i|}{x_i} = \frac{|AR_i|}{x_i}$$

(2)

A related measure is MER (Magnitude of Error Relative to the estimate [10]):

$$MER_i = \frac{|x_i - \hat{x}_i|}{\hat{x}_i} = \frac{|AR_i|}{\hat{x}_i}$$

(3)

The overall average error of MRE can be derived as the Mean or Median Magnitude of Relative Error measure (MMRE, or MdMRE respectively), can be calculated as:

$$MMRE = \frac{\sum_{i=1}^{n} MRE_i}{n}$$

$$MdMRE = median(\text{all } MRE_i)$$

(4)

(5)

A common alternative to MMRE is PRED(25), and defined as the percentage of predictions failing within 25% of the actual values, and can be expressed as:

$$PRED(25) = \frac{100}{N} \sum_{i=1}^{N} \left\{ \begin{array}{l} 1 \text{ if } MRE_i \leq \frac{25}{100} \\ 0 \text{ otherwise} \end{array} \right. $$

(6)

For example, PRED(25)=50% implies that half of the estimates are failing within 25% of the actual values [33].

There are many other performance measures including Mean Balanced Relative Error (MBRE) and the Mean Inverted Balanced Relative Error (MIBRE) studied by Foss et al. [10]:

$$MBRE_i = \frac{\hat{x}_i - x_i}{\min(\hat{x}_i, x_i)}$$

(7)

$$MIBRE_i = \frac{\hat{x}_i - x_i}{\max(\hat{x}_i, x_i)}$$

(8)
4.2 Ten Pre-processors

In this study, we investigate:

- Three simple preprocessors: none, norm, and log;
- One feature synthesis methods called PCA;
- Two feature selection methods: SFS (sequential forward selection) and SWreg;
- Four discretization methods: divided on equal frequency/width.

None is the simplest preprocessor- all values are unchanged.

With the norm preprocessor, numeric values are normalized to a 0-1 interval using Equation 9. Normalization means that no variable has a greater influence that any other.

\[
\text{normalizedValue} = \frac{(\text{actualValue} - \text{min(allValues)})}{(\text{max(allValues)} - \text{min(allValues)})}
\]

With the log preprocessor, all numerics are replaced with their logarithm. This log procedure minimizes the effects of the occasional very large numeric value.

Principal component analysis [1], or PCA, is a feature synthesis preprocessor that converts a number of possibly correlated variables into a smaller number of uncorrelated variables called components. The first component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.

Some of the preprocessors aim at finding a subset of all features according to certain criteria such as SFS (sequential forward selection) and SWR (stepwise regression). SFS adds features into an initially empty set until no improvement is possible with the addition of another feature. When ever the selected feature set is enlarged, some oracle is called to assess the value of that set of features. In this study, we used the MATLAB, objective function (which reports the the mean-squared-error of a simple linear regression on the training set). One caution to be made here is that exhaustive search algorithms over all features can be very time consuming (2^n combinations in an n-feature dataset), therefore SFS works only in forward direction (no backtracking).

SWR adds and removes features from a multilinear model. Addition and removal is controlled by the p-value in an F-Statistic. At each step, the F-statistics for two models (models with/out one feature) are calculated. Provided that the feature was not in the model, the null hypothesis is: “Feature would have a zero coefficient in the model, when it is added”. If the null hypothesis can be rejected, then the feature is added to the model. As for the other scenario (i.e. feature is already in the model), the null hypothesis is: “Feature has a zero coefficient”. If we fail to reject the null hypothesis, then the term is removed.

Discretizers are pre-processors that maps every numeric value in a column of data into a small number of discrete values:

- width3bin: This procedure clumps the data features into 3 bins, depending on equal width of all bins see Equation 10.

\[
\text{binWidth} = \text{ceiling} \left( \frac{\text{max(allValues)} - \text{min(allValues)}}{3} \right)
\]

- width5bin: Same as width3bin except we use 5 bins.
- freq3bin: Generates 3 bins of equal population size;
- freq5bin: Same as freq3bin, only this time we have 5 bins.
4.3 Nine Learners

Based on our reading of the effort estimation literature, we identified nine commonly used learners that divide into:

- Two instance-based learners: ABE0-1NN, ABE0-5NN.
- Two iterative dichotomizers: CART(yes), CART(no).
- A neural net: NNet.
- Four regression methods: LReg, PCR, PLSR, SWReg.

**Instance-based learning** can be used for analog-based estimation. A large class of ABE algorithms was described in Figure 2. Since it is not practical to experiment with the 6000 options defined in Figure 2, we focus on two standard variants. ABE0 is our name for a very basic type of ABE that we derived from various ABE studies [15, 25, 27]. In ABE0-xNN, features are firstly normalized to 0-1 interval, then the distance between test and train instances is measured according to Euclidean distance function. x nearest neighbors are chosen from training set and finally for finding estimated value (a.k.a adaptation procedure) the median of x nearest neighbors is calculated. We explored two different x:

- **ABE0-1NN**: Only the closest analogy is used. Since the median of a single value is itself, the estimated value in ABE0-1NN is the actual effort value of the closest analogy.
- **ABE0-5NN**: The 5 closest analogies are used for adaptation.

**Iterative Dichotomizers** seek the best attribute value *splitter* that most simplifies the data that fall into the different splits. Each such splitter becomes a root of a tree. Sub-trees are generated by calling iterative dichotomization recursively on each of the splits. The CART iterative dichotomizer [7] is defined for continuous target concepts and its *splitters* strive to reduce the GINI index of the data that falls into each split. In this study, we use two variants:

- **CART (yes)**: This version prunes the generated tree using cross-validation. For each cross-val, an internal nodes is made into a leaf (thus pruning its sub-nodes). The sub-tree that resulted in the lowest error rate is returned.
- **CART (no)**: Uses the full tree (no pruning).

In **Neural Nets**, or NNet, an input layer of project details is connected to zero or more “hidden” layers which then connect to an output node (the effort prediction). The connections are weighted. If the signal arriving to a node sums to more than some threshold, the node “fires” and a weight is propagated across the network. Learning in a neural net compares the output value to the expected value, then applies some correction method to improve the edge weights (e.g. back propagation). Our NNet uses three layers.

This study also uses four **regression methods**. LReg is a simple linear regression algorithm. Given the dependent variables, this learner calculates the coefficient estimates of the independent variables. SWreg is the stepwise regression discussed above. Whereas above, SWreg was used to select features for other learners, here we use SWreg as a learner (that is, the predicted value is a regression result using the features selected by the last step of SWreg). Partial Least Squares Regression (PLSR) as well as Principal Components Regression (PCR) are algorithms that are used to model a dependent variable. While modeling an independent variable, they both construct new independent variables as linear combinations of original independent variables. However, the ways they construct the new independent variables are different. PCR generates new independent variables to explain the observed variability in the actual ones. While generating new variables the dependent variable is not
considered at all. In that respect, **PCR** is similar to selection of \( n\)-many components via **PCA** (the default value of components to select is 2, so we used it that way) and applying linear regression. **PLSR**, on the other hand, considers the independent variable and picks up the \( n\)-many of the new components (again with a default value of 2) that yield lowest error rate. Due to this particular property of **PLSR**, it usually results in a better fitting.

### 4.4 Experimental Rig

This study copied the experimental rig of a recent prominent study [24]. In their leave-one-out experiment, given \( T \) projects, then \( \forall t \in T, t \) is the test and the remaining \( T - 1 \) projects are used for training. The resulting \( T - 1 \) predictions are then used to compute our seven evaluation criteria given in Section 3.1.

To compare the performance of one algorithm versus the rest, we used a Wilcoxon non-parametric statistical hypothesis test. Wilcoxon is more robust than the Student’s \( t \)-test as it compares the sums of ranks, unlike Student’s \( t \)-test which may introduce spurious findings as a result of presence of outliers may be existed in the given datasets. Ranked statistical tests like the Wilcoxon are also useful if it is not clear that the underlying distributions are Gaussian [22].

Using the Wilcoxon test, for each dataset, the performance measures collected from each of our 90 algorithms was compared to the 89 others. This allowed us to collect win-tie-loss statistics using the algorithm of Figure 3. First, we want to check if two distributions \( i, j \) are statistically different according to the Wilcoxon test (95% confident); otherwise we increment \( \text{tie}_i \) and \( \text{tie}_j \). If the distributions are statistically different, we update \( \text{win}_i, \text{win}_j \) and \( \text{loss}_i, \text{loss}_j \) after comparing their median values.

```plaintext
if WILCOXON(\text{P}_i, \text{P}_j, 95) says they are the same then
    \text{tie}_i = \text{tie}_i + 1;
    \text{tie}_j = \text{tie}_j + 1;
else
    if better(\text{median}(\text{P}_i), \text{median}(\text{P}_j)) then
        \text{win}_i = \text{win}_i + 1
        \text{loss}_j = \text{loss}_j + 1
    else
        \text{win}_j = \text{win}_j + 1
        \text{loss}_i = \text{loss}_i + 1
end if
end if
```

Fig. 3: Comparing algorithms \((i,j)\) on performance \((\text{P}_i, \text{P}_j)\). The “better” predicate changes according to \( P \). For error measures like MRE, “better” means lower medians. However, for PRED(25), “better” means higher medians.

### 5 Results

After applying leave-one-out to all 20 data sets, the procedure of Figure 3 was repeated seven times (once for MAR, MMRE, MMER, MBRE, MIBRE, MdMRE and PRED(25)). Our
### Rank Pre-Processor Learner | Rank Pre-Processor Learner
---|---
1 | norm CART (yes) | 46 | PCA | NNet
2 | norm CART (no) | 47 | width3bin ABE0-5NN
3 | none CART (yes) | 48 | none NNet
4 | none CART (no) | 49 | width5bin SWR
5 | log CART (yes) | 50 | width5bin ABE0-1NN
6 | log CART (no) | 51 | none LReg
7 | SWR CART (yes) | 52 | width3bin ABE0-5NN
8 | SWR CART (no) | 53 | SFS NNet
9 | SFS CART (yes) | 54 | norm PLSR
10 | SFS CART (no) | 55 | freq3bin ABE0-1NN
11 | SWR ABE0-1NN | 56 | width3bin NNet
12 | log ABE0-1NN | 57 | SWR LReg
13 | SWR ABE0-5NN | 58 | norm LReg
14 | SFS ABE0-5NN | 59 | freq3bin ABE0-1NN
15 | PCA PLSR | 60 | freq3bin CART (yes)
16 | SWR PCR | 61 | freq3bin CART (no)
17 | none PLSR | 62 | PCA ABE0-1NN
18 | SFS ABE0-1NN | 63 | width3bin SWR
19 | PCA PCR | 64 | width5bin PLSR
20 | none PCR | 65 | log SWR
21 | PCA CART (yes) | 66 | log PCR
22 | PCA CART (no) | 67 | log PLSR
23 | freq3bin ABE0-5NN | 68 | width3bin ABE0-1NN
24 | SWR PLSR | 69 | width3bin ABE0-1NN
25 | SFS LReg | 70 | width5bin PCR
26 | norm ABE0-1NN | 71 | norm PCR
27 | none ABE0-1NN | 72 | width3bin PCR
28 | SFS PCR | 73 | freq3bin PCR
29 | SFS PLSR | 74 | freq3bin SWR
30 | freq3bin CART (yes) | 75 | width3bin LReg
31 | freq3bin CART (no) | 76 | freq3bin PCR
32 | width5bin CART (yes) | 77 | width5bin LReg
33 | width5bin CART (no) | 78 | freq3bin PLSR
34 | norm ABE0-5NN | 79 | freq3bin PLSR
35 | PCA SWR | 80 | log LReg
36 | none ABE0-5NN | 81 | freq3bin SWR
37 | SWR SWR | 82 | freq3bin LReg
38 | SFS SWR | 83 | width3bin NNet
39 | log ABE0-5NN | 84 | norm NNet
40 | norm SWR | 85 | width3bin NNet
41 | none SWR | 86 | log NNet
42 | freq3bin ABE0-5NN | 87 | freq3bin NNet
43 | PCA ABE0-5NN | 88 | freq3bin NNet
44 | width3bin CART (yes) | 89 | freq3bin LReg
45 | width3bin CART (no) | 90 | PCA LReg

Fig. 4: Detailed algorithm combinations, sorted by the sum of their losses seen in all performance measures and all data sets. The algorithm with fewest losses is ranked #1 and is **norm/CART**(yes). At the other end of the scale, the algorithm with the most losses is ranked #90 and is **PCA/LReg**.

Ninety algorithms were then sorted by their total number of losses over all datasets. The resulting sort order is shown in Figure 4. The algorithm, with fewest losses (**norm/CART**(yes)) was ranked #1 and the algorithm with the most losses (**PCA/LReg**) was ranked #90.

Given 89 comparisons and seven performance measures and 20 datasets, the maximum number of losses for any algorithm was $89 \times 7 \times 20 = 12,460$. Figure 5 sorts all 90 algorithms according to their total losses seen in all seven performance criteria (expressed as a percentage of 12,460). The *x*-index of that figure corresponds to the ranks of Figure 4; e.g. the top ranked method of **norm/CART**(yes) lost in nearly zero percent of our experiments.

Figure 6 tests the stability of the methods. In this plot, we check if the sort orders are changed by different performance criteria:
In Figure 6, we report the mean of maximum rank changes for each method with respect
to their ordering in Figure 4.
- Each error measure defines its own ordering of methods w.r.t. its win, loss or win –
  loss values.
- Maximum rank change is the maximum absolute difference between either of these
  orderings and the ordering of Figure 4.
- Then, mean of maximum rank changes coming from 7 performance measures gives
us Figure 6.

The sort order on the x-axis of Figure 6 was kept the same as the before. A line drawn parallel
to x-axis at \( y = 10 \) gives methods, whose mean rank change is less/more than 10. See in
Figure 6 that \( y = 10 \) line divides all methods into 3 regions: a (from method 1 to 13), b (from
method 14 to 64) and c (from method 65 to 90). Regions a and c show “high-ranked” and
“low-ranked” methods respectively. None of the methods in regions a and c exceed mean
rank change of 10, i.e. they are “stable” in high and low ranks. In region b “medium-ranked”
methods are accumulated. However, all methods in region b have mean rank changes above
10, i.e. they are “unstable” in this region. In a result consistent with prior reports on ranking
instability, the lines in each region are not exactly smooth. However, they do closely follow
the same general trends as Figure 5 and Figure ??.

Since the sort orders seen using the number of losses and mean rank changes over seven
performance criteria are mostly stable, we use them to draw Figure 7. In that figure, each
\( x,y \) position shows the results of 623 comparisons (each algorithm compared to 89 oth-
ers using seven performance measures; \( 89 \times 7 = 623 \)). The y-axis of that figure shows the
90 algorithms sorted in the rank order of Figure 4. For example, the top-ranked algorithm
norm/CART(\textit{yes}) appears at \( y=1 \); the log/ABE0-INN result appears at \( y=12 \); the log/LReg
results appear at \( y=80 \); and the worst-ranked algorithm PCA/LReg appears at \( y=90 \).

In order to discuss which learners/preprocessors are “best”, we divide Figure 7 into 3
bands of Figure 6. We reserve the lowest band from method 1 to 13 (containing the “best”
estimators) for the region where all algorithms have a mean rank change of smaller than
10. Note that algorithms in that region almost always lose less than \( \frac{1}{2} \)th of the time (i.e.
the rows \( y = 1 \) to \( y = 13 \) that are almost completely yellow in Figure 7). In the other
bands (boundaried at $y = 14$ to $y = 64$ and $y = 65$ to $y = 90$), algorithms lose much more frequently, i.e. behavior of methods in the loss percentage graph of Figure 7 are in agreement with rank change graph of Figure 6.

Figure 8 shows the spectrum of PRED(25) values across the 3 bands. As might be expect, the y-axis sort order of Figure 8 predicts for estimation accuracy. As we move over the three bands from worst to best, the PRED(25) values double (approximately), thus confirming the unique performance of algorithms in each band.

Figure 9 shows the frequency counts of preprocessors and learners grouped into the five bands:

- A “good” preprocessor/learner appears often in the lower bands (tendency towards band a). In Figure 9, CART is an example of a “good” learner.
- A “poor” preprocessor/learner appears more frequently in the higher bands (tendency towards band c). In Figure 9, all the discretization preprocessors (e.g. freqbin) are “poor” preprocessors.
- The gray rows of Figure 9 shows preprocessor/learner that are neither “good” nor “poor” (since they exist in all 3 bands have high frequency counts in bands b and c); e.g. see the log preprocessor.

6 Discussion

6.1 Findings

Based on these figures and results, we summarize our findings as follows.

Result1: Observe how the majority of the squares on the left-hand-side of Figure 7 are yellow. In that mostly-yellow region, algorithms loss vary rarely against other algorithms (in less that $\frac{1}{2}$ of all comparisons). Also notice how higher loss percentages (more than 50%) become dominant on the right-hand-side. For the purposes of finding the best effort estimator, the data sets on the far left and right-hand-sides are not very suitable since they hardly distinguish the performance of different algorithms.

Result2: Observing the small amounts of “jitter” in Figure 6 we can see that our results are not 100% stable, they are only sufficiently stable to draw conclusions. We conjecture that prior reports on ranking instability could stem from using too few data sets or too few algorithms.

Fig. 6: Algorithms and the mean of their maximum rank changes over all performance measures. Mean rank change of smaller than 10 divides 90 methods into 3 regions. Region a consists of high-ranked stable methods, whereas region contains low-ranked but still stable methods. Region b on the other hand shows middle-ranked and non-stable methods.
Fig. 7: Number of losses seen in 90 methods and 20 datasets, expressed as a percentage of the maximum losses possible for one method in one dataset (so 100% = 623; 50% = 311; 25% = 156; 12.5% = 78). The algorithms on the y-axis are sorted according to Figure 5.

![Diagram](image)

Fig. 8: Spectrum of Pred(25) across the bands

**Result3:** Observe how, in Figure 4, learners found at one rank with a one preprocessor, can jump to a very different rank if the different preprocessor is changed. For example, the top-ranked method that uses CART(yes), is driven down to rank 60 if the preprocessor is changed from norm to freq3bin. Clearly, the effectiveness of a learner can be signifi-
Table 1: Occurrence of algorithms in bands a, b, c

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Band a</th>
<th>Band b</th>
<th>Band c</th>
</tr>
</thead>
<tbody>
<tr>
<td>CART (yes)</td>
<td>34</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>CART (no)</td>
<td>33</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>ABE0-5NN</td>
<td>6</td>
<td>55</td>
<td>2</td>
</tr>
<tr>
<td>ABE0-INN</td>
<td>11</td>
<td>44</td>
<td>8</td>
</tr>
<tr>
<td>PCR</td>
<td>3</td>
<td>29</td>
<td>31</td>
</tr>
<tr>
<td>PLSR</td>
<td>3</td>
<td>35</td>
<td>25</td>
</tr>
<tr>
<td>LReg</td>
<td>22</td>
<td>22</td>
<td>41</td>
</tr>
<tr>
<td>SWR</td>
<td>46</td>
<td>17</td>
<td>26</td>
</tr>
<tr>
<td>NNet</td>
<td>20</td>
<td>43</td>
<td>10</td>
</tr>
</tbody>
</table>

The relatively poor performance of simple linear regression is a highly significant result. **LReg**, with a log preprocessor, is the core technology of many prior publications; e.g. the entire COCOMO project [5]. Yet as shown in Figure 7, w.r.t. loss values over all error measures, **log/LReg** ranks very poorly (position 80 out of a maximum of 90 algorithms). We also did experiments at individual level of error measures. At individual level the ranking is not very different either, i.e. the ranking of LReg w.r.t. loss values over MAR, MMRE, MMER, MBRE, MIBRE, MdMRE and Pred(25) are 80, 76, 81, 80, 75, 76 and 78 respectively.

Result 5: While **SWR** falls into the worst two bands of the learners, it is most commonly found in the best two bands of the preprocessors. That is, stepwise regression is a poor learner but a good preprocessor. Hence, in future, the fate of **SWR** might be as an assistant to other algorithms.

Result 6: While simple regression methods like **LReg** are depreciated by this study, more intricate regression methods like regression trees (CART) and partial linear regression **PLSR** are found in the better bands. Hence, in future, proponents of regression for effort estimation might elect to explore more intricate forms of regression than just simple **LReg**.

Result 7: The top-ranked algorithm was **norm/CART**.

Result 8: Simple methods (e.g. k=1 nearest neighbor on the log of the numerics) perform nearly as well as the top-ranked algorithm. Figure 10 compares the PRED(25) results between rank=12 and rank=1. The datasets in that figure are sorted by the difference between the top-ranked and the twelfth-ranked algorithm. Except for China dataset, the difference in PRED(25) values is either slightly negative, or positive. That is, even though the rank=1 al-
Algorithm is relatively “best” (measured according to our comparative Wilcoxon tests), when measured in an absolute sense, it is not impressively better than simpler alternatives.

<table>
<thead>
<tr>
<th></th>
<th>norm/CART (yes)</th>
<th>log/ABE0-1NN</th>
<th>difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>kemeter</td>
<td>7</td>
<td>24</td>
<td>-17</td>
</tr>
<tr>
<td>desharnaisL3</td>
<td>20</td>
<td>40</td>
<td>-20</td>
</tr>
<tr>
<td>nas93_center_2</td>
<td>43</td>
<td>57</td>
<td>-14</td>
</tr>
<tr>
<td>nas93</td>
<td>29</td>
<td>39</td>
<td>-10</td>
</tr>
<tr>
<td>cocomo81s</td>
<td>9</td>
<td>18</td>
<td>-9</td>
</tr>
<tr>
<td>albrecht</td>
<td>33</td>
<td>42</td>
<td>-9</td>
</tr>
<tr>
<td>telecom1</td>
<td>33</td>
<td>39</td>
<td>-6</td>
</tr>
<tr>
<td>cocomo811</td>
<td>13</td>
<td>16</td>
<td>-3</td>
</tr>
<tr>
<td>nas93_center_5</td>
<td>36</td>
<td>33</td>
<td>3</td>
</tr>
<tr>
<td>desharnaisL3</td>
<td>39</td>
<td>35</td>
<td>4</td>
</tr>
<tr>
<td>cocomo810</td>
<td>29</td>
<td>21</td>
<td>8</td>
</tr>
<tr>
<td>desharnaisL2</td>
<td>48</td>
<td>40</td>
<td>8</td>
</tr>
<tr>
<td>cocomo81e</td>
<td>18</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>desharnais</td>
<td>43</td>
<td>32</td>
<td>11</td>
</tr>
<tr>
<td>sdr</td>
<td>42</td>
<td>29</td>
<td>13</td>
</tr>
<tr>
<td>miyazaki94</td>
<td>40</td>
<td>25</td>
<td>15</td>
</tr>
<tr>
<td>maxwell</td>
<td>32</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>finnish</td>
<td>61</td>
<td>37</td>
<td>24</td>
</tr>
<tr>
<td>nas93_center_1</td>
<td>58</td>
<td>33</td>
<td>25</td>
</tr>
<tr>
<td>china</td>
<td>95</td>
<td>43</td>
<td>52</td>
</tr>
</tbody>
</table>

Fig. 10: Using all data sets to compare the Pred(25) of norm/CART (rank=1) and log/ABE0-1NN (rank=12).

Result 8 is an important result, for three reasons. Firstly, there are many claims in the literature that software project follows a particular parametric form. For example, in the COCOMO project, that form is \( \text{effort} \propto KLOC^n \). The fact that non-parametric instance methods perform nearly as well as our best method suggests that debates about the parametric form of effort estimation is misguided. Also, it means that the value of certain commercial estimation tools based on a particular parametric form may not be much more than simple instance-based learners.

Secondly, analogy-based estimation methods are widely used [2, 16–18, 20, 23–25, 33–35]. Result 8 says that while this approach may not be 100% optimal in all cases, compared to our best estimator found by this study, there is not a dramatic lost if estimates are generated by analogy. Prior to this publication, we are unaware of a large comparative study relating to this matter.

Thirdly it is easier to teach and experiment with simpler algorithms (like the log/ABE0-1NN algorithm at rank=12) than more complex algorithms (like the norm/CART algorithm at rank=1). For example, recently we have been experimenting with a very simple variant of ABE0-1NN that is useful as a learner to find software process change [6]. Such experimentation would have been hindered if we tried to modify the more complex CART algorithm (particularly if we included sub-tree pruning).

6.2 Validity

*Construct validity* (i.e. face validity) assures that we are measuring what we actually intended to measure [31]. Previous studies have concerned themselves with the construct validity of different performance measures for effort estimation (e.g. [10]). While, in theory, these performance measures have an impact on the rankings of effort estimation algorithms,
we have found that other factors dominate. For example, Figure 7 showed that some of the
datasets have a major impact on what could be concluded after studying a particular estimator
on these data set. We also show empirically the surprising result that our results regarding
algorithms are stable across a range of performance criteria.

External validity is the ability to generalize results outside the specifications of that
study [29]. To ensure external validity, this paper has studied a large number of projects.
Our data sets are diverse, measured in terms of their sources, their domains and the time
they were developed in. We use datasets composed of software development projects from
different organizations around the world to generalize our results [4]. Our reading of the
literature is that this study uses more data, from more sources, than numerous other papers.
For example, Table 4 of [21] list the total number of projects used by a sample of other
studies. The median value of that sample is 186; i.e. one-sixth of the 1198 projects used
here.

As to the external validity of our choice of algorithms, recalling Figure 2, it is clear
that this study has not explored the full range of effort estimation algorithms. Clearly, future
work is required to repeat this study using the “best of breed” found here (e.g. bands one
and two of Figure 9 as well as other algorithms).

Having cast doubts on our selection of algorithms, we hasten to add that this paper has
focused on algorithms that have been extensively studied in the literature [33] as well as
the commonly available datasets (that is, the ones available in the PROMISE repository
of reusable SE data). That is, we assert that these results should apply to much to current
published literature on effort estimation.

7 Conclusion

In this study, ten learners and nine data preprocessors were combined into 90 effort estimation
algorithms. These were applied to twenty datasets. Performance was measured using
seven performance indicators (AR, MRE, MER, MdMRE, MMRE, PRED(25), MBIRE).
Performances were compared using a Wilcoxon ranked test (95%). To the best of our know-
ledge, this is the largest and most comprehensive effort estimation study yet reported in the
literature. Eight results are noteworthy:

1. Prior reports of ranking instability about effort estimation may have been overly pes-
simistic. Given relatively large number of publicly available effort estimation datasets,
it is now possible to make stable rankings about the relative value of different effort
estimators.
2. The effectiveness of a learner used for effort estimation (e.g. regression or analogy meth-
ods) can be significantly altered by data preprocessing (e.g. logging all numbers or nor-
malizing them zero to one).
3. Neural nets and simple linear regression perform much worse than other learners such
as analogy learners.
4. Stepwise regression was a very useful preprocessor, but surprisingly a poor learner.
5. Non-simple regression methods such as regression trees and partial linear regression are
relatively strong performers.
6. Regression trees that use tree pruning performed best of all in this study (with a prepro-
cessor that normalized the numerics into the range zero to one).
7. Very simple methods (e.g. K=1 nearest neighbor on the log of the numerics) performed
nearly as well as regression trees.
Lastly, this is an empirical paper that reports, but does not explain, the rankings of data sets and algorithms seen in Figure 7. An open question raised by this work is what features of our algorithms resulted in their rankings. While we have no current theory on what explains the algorithm ordering, we speculate that the dataset ordering might be explained by the regions of local high variance in their internal structure. However, at the time of this writing, we have no convincing evidence for that speculation.

Given the significance of this study, an important goal for future work would be to determine the reason for the algorithm ranking seen in this study.

References


