Effort Estimation for Solo methods *

Anbarasi Manoharan
North Carolina State University
amanoha2@ncsu.edu

Mathioli Ramalingam
North Carolina State University
mramali2@ncsu.edu

ABSTRACT

Background: Despite decades of research, there is no consensus on which software effort estimation methods produce the most accurate models. ***Aim:*** Prior work has reported that, given M estimation methods, no single method consistently outperforms all others. Perhaps rather than recommending one estimation method as best, it is wiser to generate estimates from ensembles of multiple estimation methods. ***Method:*** 6 learners were combined with 8 preprocessing options to generate 6 X 8 = 48 solo methods. These were applied to 10 datasets and evaluated using seven error measures. We rank the solos based on each error measure across all datasets, based on each dataset across all error measures and also ranked solos across all datasets and error measures. ***Results:*** Solos cannot be better ranked on frequency or cumulative methodology alone. Both the methods has to be combined, to rank the solos better. Conclusion: There is no best single effort estimation method.

Keywords

Software cost estimation, machine learning, regression trees, principal component analysis, analogy, k-NN

1. INTRODUCTION

Estimating the effort required to develop software is always challenging. Over or under estimation of effort can lead to unfavorable results. If one over estimates the effort, it might affect the project being over quoted, thus losing the project to the competitor. If one under estimates the effort, it might lead to promise being given which will eventually gets failed or working over time, which may cause project cancellation or losing client's trust.

Some studies suggest that it may be impossible to assess which effort estimators are the best [7]. Shepperd et al. [17] warn that when comparing M estimation methods, if the conditions are changed, the ranking of any one method may change. Hence, they argue that it is fundamentally impossible to offer a definitve ranking such that one method is better than other method.

Menzies et al. [11] suggests that if we combine the estimates from multiple estimators, then those combined methods perform better than any single estimator. Researchers in machine learning agree averaging the estimators from many methods often does better than any solo method [16]. Previous studies on the ensembles report that the ensembles are not statistically better than single learners, but Menzies et al. [11] reports that ensembles can outperform single learners.

We tried to re-do part of the work done by Menzies et al. [11] by building solo methods and running across multiple datasets. Then all the solo methods are sorted based on its rank among all the datasets.

2. EFFORT ESTIMATION METHODS

The effort estimation methods in this paper is a solo methods, which is a combination of *preprocessing option* and a *learner*. From now on *"learner"* refers to a stand-alone algorithm without any preprocessing option embedded (e.g., CART, KNN, etc.), and *"Solo methods"* refers to an algorithm followed by a preprocessing option (e.g., CART + log, KNN + norm, etc.).

2.1 Preprocessing options

2.1.1 Simple numeric techniques

. This category lists the preprocessors that entail mere numeric alterations of actual values rather than application of certain algorithms.

- norm. "norm" represents the application of normalization on the data. The data are normalized to 0-1 interval according to (1).

$$normalizedValue = \frac{actual - min(allValues)}{max(allValues) - min(allValues)} \quad (1)$$

- log. Take the natural logarithm of the independent variables.

2.1.2 Feature synthesis

"PCA" stands for principal component analysis [3]. PCA is a dimension alteration mechanism, where an n-dimensional space is altered into another n-dimensional space.

2.1.3 Discretization

- **width3bin.** This procedure bins each one of the data features into three bins, depending on equal width of all bins. The bin-width for a general n-bin procedure is given in (10). In our 3-bin case, once we know the bin-width, we assign each feature value to either 1, 2, or 3, depending on which particular bin the value is in (2)

*Rework of [11]

$$binWidth = ceil(\frac{max(allValues) - min(allValues)}{n}) \quad (2)$$

- **width5bin.** Exactly same as "width3bin" except that this time we have five bins instead of 3
- **freq3bin.** "freq3bin" means binning each feature into three bins, depending on the equal frequency count (equal number of instances in each bin). Similar to previously mentioned binning methods, in this method each feature value is assigned to 1, 2, or 3. For that, all values of a particular feature are sorted in ascending order. Then, first ceiling(numberOf (allInstances)/3) instances are assigned 1, the second ceiling(numberOf (allInstances)/3) instances are assigned 2, and so on.
- **freq5bin.** The same as "freq3bin" but with five bins.

2.1.4 Others

The option(s) that cannot be categorized into the aforementioned categories are mentioned here. Application of no preprocessing to the data is also a choice. "none" represents the choice of no preprocessor selection.

2.2 Learners

2.2.1 Instance-based learners

In ABE, effort estimates are generated for a test project by finding similar completed software projects (a.k.a. the training projects). Following Menzies et al. [10] we define a baseline ABE called ABE0, as follows: ABE0 executes over a table of data where

- each row contains one project;
- columns contain independent variables (features) in the projects and dependent variables (features) that store, for example, effort and duration required to complete one project.

After processing the training projects, ABE0 inputs one test project then outputs an estimate for that project. To generate that estimate, a scaling measure is used to ensure all independent features have the same degree of influence on the distance measure between test and training projects. Also, a feature weighting scheme is applied to remove the influence of the less informative independent features. For example, in feature subset selection, some features are multiplied by zero to remove redundant or noisy features. The similarity between the target project case and each case in the case-based repository is determined by a similarity measure. There are different methods of measuring similarity that have been proposed for different measurement contexts. A similarity measure is measuring the closeness or the distance between two data objects in an n-dimensional feature space; the result is usually presented in a distance matrix (or similarity matrix) identifying the similarity among all cases in the data set. The euclidean distance metric is the most commonly used in ABE for its distance measures, and it is suitable for continuous values such as software size, effort, and duration of a project. It is based on the principle of the Pythagorean Theorem to derive a straight line distance between two points in n-dimensional space. In general, the unweighted euclidean distance between two points P =(p_1, p_2, ... , p_n) and Q = (q_1, q_2, ... , q_n), and can be defined and calculated as (3)

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \quad (3)$$

An alternative is to apply different weights to each individual project feature to reflect its relative importance in the prediction system. The weighted euclidean distance can be calculated as (4)

$$\sqrt{w_1(p_1 - q_1)^2 + w_2(p_2 - q_2)^2 + ... + w_n(p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} w_i(p_i - q_i)^2} \quad (4)$$

where w_1 and w_n are the weights of the first and nth project features. Note that in the special case of $w_i = 1$ (i.e., equal weighting) the equations are identical. The above euclidean distance functions are suitable for general problems, particularly when values are of continuous nature. There are other different distance metrics for noncontinuous variable; these include, but are not limited to, Jaccard distance for binary distance and Gower distance, described by Gower and Legendre. In this paper, we only consider the euclidean distance measure, which is most relevant to the context of software cost estimation. Irrespective of the similarity measure used, the objective is to rank similar cases from the data set to the target case and utilize the known solution of the nearest k cases. The value of k in this case has been the subject of debate in the ABE research community. Shepperd and Schofield suggested the ideal value for k is 3, that is, only three closest neighboring cases will be considered. These k cases will be adjusted or adapted to better fit the target problem by predefined rules, a human expert, or more commonly, using a simple mean or median of the selected k cases.

2.2.2 Alternatives to ABE0

The analogy-based estimation method that we call ABE0-xNN refers to a very basic type of ABE that we defined through our readings of various ABE studies [14], [13], [8]. In ABE0-xNN, features are first normalized to the 0-1 interval, then the distance between the test and train instances is measured according to euclidean distance function, x nearest neighbors are chosen from the training set, and finally, for finding estimated value (a.k.a adaptation procedure), the median of x nearest neighbors is calculated. Therefore, when we say ABE0- xNN, all the design options including the choice of x-many closest analogies become explicit to a reader. In our experimentation we use two different x values (i.e., two different analogy values):

- ABE0-1NN: Only the closest analogy is used.
- ABE0-5NN: The five closest analogies are found and used for adaptation.

2.2.3 Iterative dichotomization

Table 1: Summary of Solo Methods

Pre-processing Options		Learners	
Abbreviation	**Explanation**	**Abbreviation**	**Explanation**
norm	Normalization	CART (no)	Classification and Regression Tree without pruning
log	Natural logarithm	ABE0-1NN	Basic ABE with 1 nearest neighbor
PCA	Principal Component Analysis	ABE0-5NN	Basic ABE with 5 nearest neighbor
freq5bin	Discretize into 5 bins based on equal frequency	PLSR	Partial Least Square Regression
freq3bin	Discretize into 3 bins based on equal frequency	PCR	Principal Component Regression
width5bin	Discretize into 5 bins based on equal width	LReg	Simple linear regression
width3bin	Discretize into 3 bins based on equal width		
none	No pre-processor		

Under this category, we used classification and regression trees (CART) as described by Breiman et al. [6] and developed it using Matlab routines. CART is a nonparametric technique and it can work both for classification and regression type of problems; in our case it is the latter. When planning to construct a CART, there are a couple of points to consider. First is the selection of splits: There are a variety of solutions such as misclassification rate, information (or entropy), or GINI index. CART uses the GINI index to calculate the impurity of a tree [6]. Second, the decision on when to stop further splits: The implementation allows you to specify a threshold value or use the default value. We have used default threshold values in the implementation (for further details please refer to Matlab documentation). Last, assigning a class to each terminal node. Each test instance results in a terminal node and therefore different test instances resulting in the same terminal node are given the same predicted value. For regression, the predicted value in a terminal node is a fit on the independent variables of the instances in that node. We have used CART (no). **CART (no)** The subtrees of CART will not be considered and the full tree will be used.

2.2.4 Regression methods

Under this category, we list our regression-based learners. **LReg** stands for linear regression. Given the dependent variables, this learner calculates the coefficient estimates of the independent variables. **Partial least squares regression (PLSR)** as well as **principal components regression (PCR)** are algorithms that are used to model a dependent variable and we have used Matlab routines to implement those functions in our experiments. While modeling an independent variable, they both construct new independent variables as linear combinations of original independent variables. However, the ways in which they construct the new independent variables are different:

- **PCR** This method generates new independent variables to explain the observed variability in the actual ones. However, 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** Unlike PCR, PLSR 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.

- **Linear Regression** There are many induction methods like linear regression, neural networks and analogy based estimators. Linear regression is useful when the data fits some function. The parameters of the function are then adjusted in order to reduce the difference between the actual and predicted effort estimations. An example of a linear regression model is :

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...$$

  where x_i are model input features, y is the model output and β_i are the coefficients/model parameters adjusted be the linear regression system. The issue with linear regression is that it assumes the data is linearly separable. But this is not possible in all the cases. Various patches have been proposed where we take logarithms of exponential distributions before applying linear regression.

2.3 Solo Methods

In our paper we have used 8 different preprocessing options and 6 learners. They were selected on two factors, one it should be from effort estimation literature and other is that learners should make different assumptions about data, as different learners may fail under different kinds of data. For example, ABE methods assume that similar instances of the dataset have similar dependent variable values whose translation into SEE domain is that similar projects have similar effort values [19]. Decision tree inducers adopt a divide and conquer approach and assume that decisions on the prediction can be made through a sequence of tests/decisions that usually involve one feature at a time [20]. Linear regression assumes that the trend seen in data can be formulated through a linear model [15].

We used 8 preprocessing options:

- three *simple preprocessing* options: none, norm, and log,

- one *feature synthesis* method: PCA,

- four *discretization* methods: Based on equal frequency/ width

and 6 learners:

- one *Iterative dichotomizer*: CART(no),

- three *Regression methods*: LReg, PCR, PLSR,
- two *instance-based learners*: ABE0-1NN, ABE0-5NN.

Note that "ABE" is short for analogy-based effort estimation. ABE0-kNN is a standard ABE with execution steps of

- normalization of data to zero-one interval,
- a euclidean distance measure,
- estimates generated using the k nearest neighbors.

For a brief list of the learners and preprocessors, see Table 1; for their detailed descriptions, see the Appendix.

We have combined these 8 preprocessing options with 6 learners, and built 8 X 6 = 48 solo methods. We have analyzed these solo methods and ranked them on various categories.

- Ranking based on the frequency of it being ranked 1 by Scott-Knott, and
- Cumulative ranking.

We have included these results at Table 3 and 2 respectively. Solo method **norm PCR** means that dataset has to be normalized and then PCR learner should be applied in the dataset. These solo methods has to be run only once, to identify the better performing solo methods. Once better performing solos are identified, we can use it to build multimethods, which is a combination of solo methods.

2.4 Cross Validation

In k-fold cross-validation, sometimes called rotation estimation, the dataset D is randomly split into k mutually exclusive subsets (the folds) D_1, D_2, ..., D_k, it is trained on $\frac{D}{D_t}$ and tested on D_t. The cross-validation estimate of accuracy is the overall number of correct classifications, divided by the number of instances in the dataset. Formally, let D_i be the test set that includes instance $x_i = (v_i, y_i)$, then the cross-validation estimate of accuracy,

$$acc_{cv} = \frac{1}{n} \sum_{(v_i, y_i) \epsilon D} \delta(x(\frac{D}{D_i, v_i}), y_i) \quad (5)$$

The cross-validation estimate is a random number that depends on the division into folds. Complete cross-validation is the average of all possibilities for choosing $\frac{m}{k}$ instances out of m, but it is usually too expensive. Except for leave-one-one (n-fold cross-validation), which is always complete, k-fold cross-validation is estimating complete k-fold cross-validation using a single split of the data into the folds [12]. Repeating cross-validation multiple times using different splits into folds provides a better Monte-Carlo estimate to the complete cross-validation at an added cost. In stratified cross-validation, the folds are stratified so that they contain approximately the same proportions of labels as the original dataset.

In 5-way cross-validation, the dataset T is divided into 5 bins. When a bin_i is used for testing, the remaining $T - bin_i$ bins are used for training. Even though we used the 5x5 cross-validation, the package also supports Leave-One-Out cross-validation.

2.5 Scott-Knott

The procedure begins by participating the groups to maximize the sum of squares between groups. The process is facilitated when the means are ordered, since the number of possible partitions (g-1 partitions) is reduced.

The sum of squares is defined as B_0, according to the expression:

$$B_0 = \frac{T_1^2}{k_1} + \frac{T_2^2}{K_2} - \frac{(T_1 + T_2)^2}{K_1 + K_2} \quad (6)$$

Where T_1 and T_2 are the totals of the two groups with K_1 and K_2 treatments in each.

The values obtained are tested by the statistics 1 according to the expression:

$$\lambda = \frac{\pi}{2(\pi - 2)} x \frac{B_0}{\hat{\sigma}^2} \quad (7)$$

where $\hat{\sigma_0}^2$ is the estimator of maximum likelihood $\sigma^{\frac{2}{y}}$ obtained by:

$$\sigma_0^2 = \frac{1}{g + v} [\sum_{i=1}^{x} (Y_i - Y)^2 + v s_y^2] \quad (8)$$

where Y_i: mean of treatment i (i=1,2,...g); Y: overall mean of treatments to be seperated; g: number of means to be seperated; v: number of residual degrees of freedom; s_y^2: QMR/r being r the number of observations that created the means to be grouped.

The statistics 1 is tested by the chi square statistic (X^2), where 1 $\lambda < X^2(\alpha, v_0)$ indicates that the two groups are statistically different and should be tested seperately for new possible divisions [4].

For example, we have set 1 to 10. 1, 2, 3, 4, 5, 6, 7 and 8 is divided into one group and second group is 9 and 10. The next step consists in the attempt to partition the formed groups again. The first group was once again divided into two subgroups (one with 1, 2, 3, 4, 5 and the other with 6, 7, and 8). In these newly formed groups, new possible partitions were sought. The statistics showed that the two groups could not be partitioned. The group composed of 9 and 10 was also divided into two subgroups with one each. This way, all statistically possible partitions were performed, forming homogeneous subgroups. The final result obtained by the test Scott-Knott is shown in Figure **??**.

3. METHODOLOGY

3.1 Multiple Error Measures

This section describes several performance measures used in this research. All the performance measures listed here has been used for effort estimation before (Refer Table 4).

Error measures comment on the success of a prediction. For example, the absolute residual (AR) is the difference between the predicted and the actual values as given in the equation (9) where x_i and $_i$ is the actual and the predicted value

$$AR_i = |x_i - \hat{x}_i| \quad (9)$$

MAR is the mean of individual AR values.

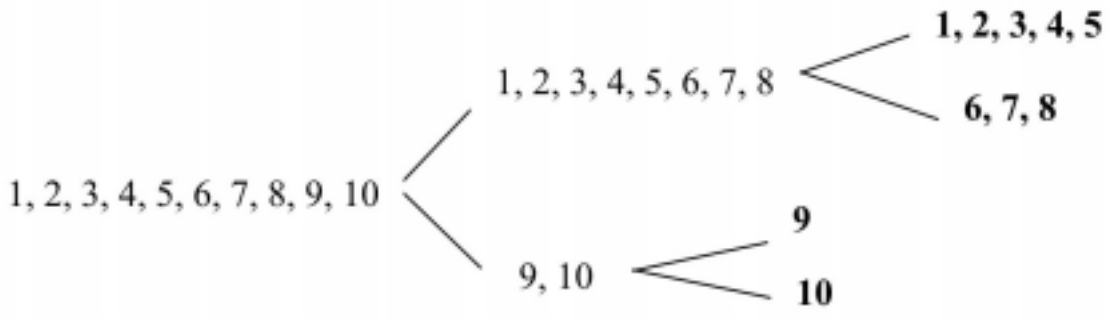

Figure 1: Partitions performed by the Scott-Knott at 5% probability

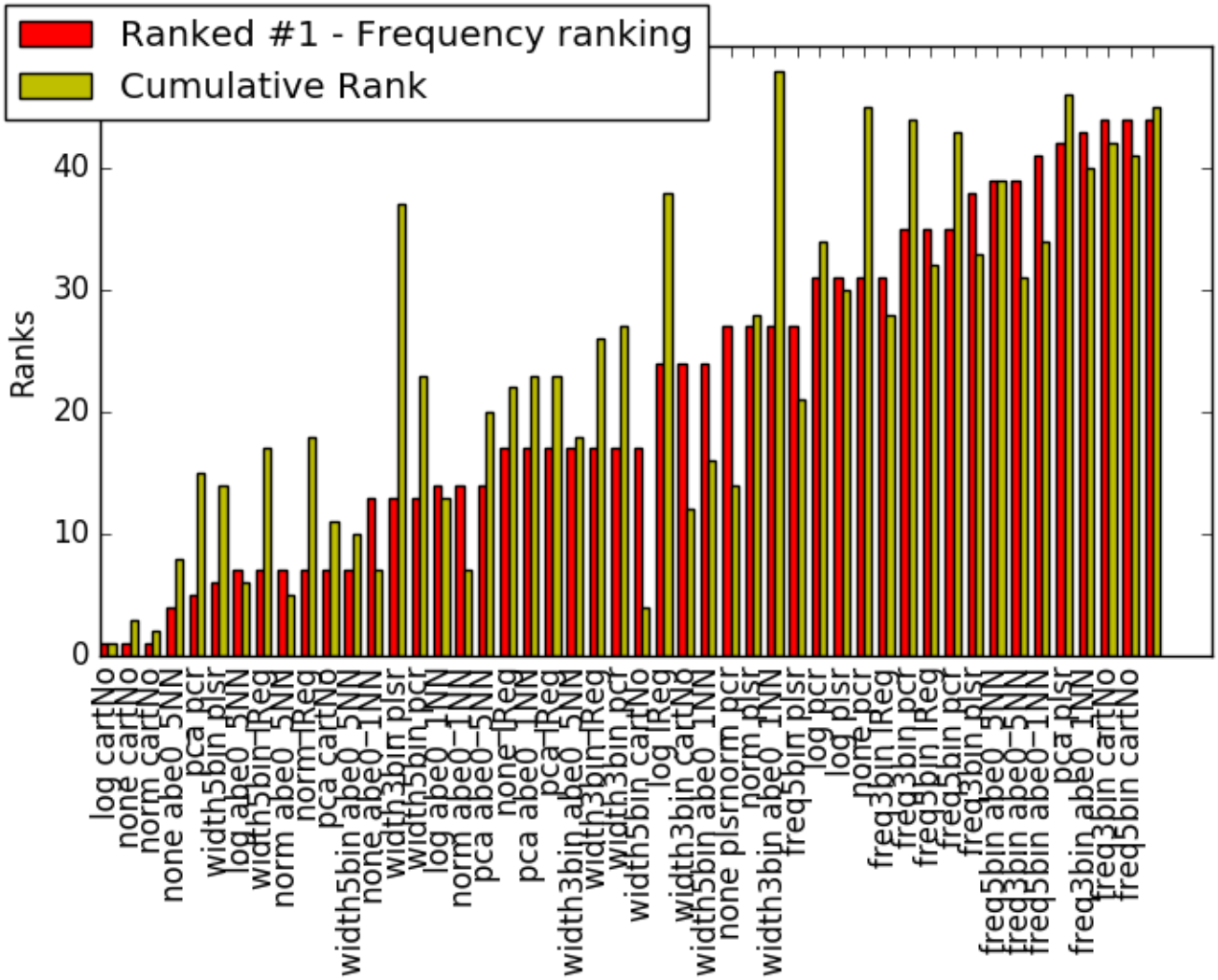

Figure 2: Comparision of the two ranking methods

Table 2: Cumulative Ranking

Rank	Cumulative Ranking from Scott-Knott	Solo
1	138	log + Cart (no)
2	139	norm + Cart (no)
3	143	none + Cart (no)
4	150	width5bin + Cart (no)
5	151	norm + ABE0_5NN
6	152	log + ABE0_5NN
7	153	none + ABE0_1NN
7	153	none + ABE0_5NN
7	153	norm + ABE0_1NN
10	156	width5bin + ABE0_5NN
11	157	PCA + Cart (no)
11	157	width3bin + Cart (no)
13	159	log + ABE0_1NN
14	161	width5bin + PLSR
15	165	PCA + PCR
15	165	width5bin + ABE0_1NN
17	170	width5bin + lReg
18	171	norm + lReg
18	171	width3bin + ABE0_5NN
20	172	PCA + ABE0_5NN
20	172	width3bin + ABE0_1NN
21	176	none + lReg
22	177	PCA + ABE0_1NN
22	177	PCA + lReg
22	177	width5bin + PCR
25	182	width3bin + lReg
26	189	width3bin + PCR
27	195	none + PCR
28	195	norm + PCR
29	214	log + PCR
30	220	freq5bin + ABE0_5NN
31	228	freq3bin + PCR
32	229	freq5bin + PCR
33	232	freq5bin + ABE0_1NN
33	232	freq5bin + PLSR
35	236	freq3bin + ABE0_5NN
35	236	width3bin + PLSR
37	238	log + lReg
38	244	freq3bin + PLSR
39	252	freq3bin + ABE0_1NN
40	253	freq5bin + Cart (no)
41	254	freq3bin + Cart (no)
42	259	freq5bin + lReg
43	260	freq3bin + lReg
44	268	log + PLSR
45	299	PCA + PLSR
46	308	none + PLSR
47	308	norm + PLSR

Sorted based on the cumulative rank of 10 datasets, calculated by Scott-Knott.

Table 3: Frequency of solos being ranked 1

Rank	Frequency of Rank1 by Scott-Knott	Solo
1	42	log + Cart (no)
1	42	none + Cart (no)
1	42	norm + cart (no)
4	40	none + ABE0_5NN
5	39	PCA + PCR
6	37	width5bin + PLSR
7	36	log + ABE0_5NN
7	36	width5bin + LReg
9	35	norm + ABE0_5NN
9	35	norm + LReg
9	35	PCA + Cart (no)
9	35	width5bin + ABE0_5NN
13	34	none + ABE0_1NN
13	34	width3bin + PLSR
13	34	width5bin + PCR
16	33	log + ABE0_1NN
16	33	norm + ABE0_1NN
16	33	PCA + ABE0_5NN
19	32	none + lReg
19	32	PCA + ABE0_1NN
19	32	PCA + lReg
19	32	width3bin + ABE0_5NN
19	32	width3bin + lReg
19	32	width3bin + PCR
19	32	width5bin + Cart (no)
26	31	log + lReg
26	31	width3bin + Cart (no)
26	31	width5bin + ABE0_1NN
29	30	none + PLSR
29	30	norm + PCR
29	30	norm + PLSR
29	30	width3bin + ABE0_1NN
33	29	freq5bin + PLSR
33	29	log + PCR
33	29	log + PLSR
33	29	none + PCR
37	28	freq3bin + lReg
37	28	freq3bin + PCR
37	28	freq5bin + lReg
40	27	freq5bin + PCR
41	26	freq3bin + PLSR
41	26	freq5bin + ABE0_5NN
43	25	freq3bin + ABE0_5NN
44	24	freq5bin + ABE0_1NN
45	24	PCA + PLSR
46	23	freq3bin + ABE0_1NN
47	23	freq3bin + Cart (no)
48	23	freq5bin + Cart (no)

Sorted based on the frequency of solos being ranked 1, among 10 datasets, by Scott-Knott.

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. MRE measures the error ratio between the actual effort and the predicted effort. It can be expressed as the equation (10)

$$MRE_i = \frac{|x_i - \hat{x}_i|}{x_i} = \frac{AR_i}{x_i} \tag{10}$$

A related measure is MER (Magnitude of Error Relative to the estimate) found at equation (11)

$$MER_i = \frac{|x_i - \hat{x}_i|}{\hat{x}_i} = \frac{AR_i}{\hat{x}_i} \tag{11}$$

A summary of MRE can be derived as the Mean Magnitude of Relative Error (MMRE) or Median Magnitude of Relative Error (MdMRE), which can be calculated as given in the equations (12) and (13)

$$MMRE = \frac{\sum_{i=1}^{n} MRE_i}{n} \tag{12}$$

$$MdMRE = median(MRE_1, MRE_2, ..MRE_n) \tag{13}$$

A common alternative error measure is PRED(25), which can be defined as the percentage of predictions falling within 25 percent of the actual values as given in the equation (14)

$$PRED(25) = \frac{100}{N} \sum_{i=1}^{N} \begin{cases} 1, if MRE_i \leq \frac{25}{100} \\ 0, otherwise \end{cases} \tag{14}$$

For example, PRED(25)= 50% implies that half of the estimates fall within 25 percent of the actual values.

There are many other error measures including Mean Balanced Relative Error (MBRE) and the Mean Inverted Balanced Relative Error (MIBRE) which are the means of BRE_i and $IBRE_i$ as shown in equation (15) and equation (16)

$$BRE_i = \frac{|x_i - \hat{x}_i|}{min(x_i, \hat{x}_i)} \tag{15}$$

$$IBRE_i = \frac{|x_i - \hat{x}_i|}{max(x_i, \hat{x}_i)} \tag{16}$$

3.2 Experimental Conditions

Different experimental conditions can change the rank of an effort estimator [18]. Hence, it is important to study not just the rank of an estimator, but also how well that performs across various conditions such as, error measures that measure a method's performance; comparision summaries; and datasets used in the experiments.

The *error measures* used in this study, as defined above, are AR, MRE, MER, MMRE, MdMRE, PRED(25), BRE, and IBRE.

As to *comparision summaries*, we have used two different methods to calculate the rank of the Solos. One, based on the frequency of the solo being ranked 1 and the other, based on cumulative ranking.

Ranking based on the frequency of a solo being ranked 1 by Scott-Knott The error measures for all the solos, for every dataset is calculated, and then the results are sent to Scott-Knott. Scott-Knott will return a ranked list of solos based on that particular error measure for that dataset. After getting the results from Scott-Knott, we are calculating the frequency of each solo being ranked 1 and we are ranking all the solo methods, based on this frequency. We are using Standard Competition ranking, to rank the solos, i.e., solos that compare equal are both ranked with same number, and the gap is left in the ranking numbers. For example, if one solo ($method_1$) is ranked 1 in six out of eight outcomes from Scott-Knott and, the other solo ($method_2$) is ranked 1 in two out of eight outcomes, then we say $method_1$ is better than $method_2$ and $method_1$ is ranked higher than $method_2$.

In Table 3,We can notice that the solo log + Cart (no) is ranked 1 here, as it has been ranked 1 by Scott-Knott the most number of times among all the solos.

Cumulative Ranking The ranking, for all the solos, provided by the Scott-Knott are recorded. Then the rank of individual solos, for every error measure, for every dataset are added (cumulative ranking). The solo methods are sorted based on this cumulative value and are ranked. We are using Standard Competition ranking, for this as well.

Finally, we are using 10 publicly available effort estimation *datasets* in the PROMISE repository. The names and properties of the dataset are provided in Table 5. Some datasets contain missing values and we used median imputation [1] to handle missing values. See the Appendix for more explanation.

Symbol	Explanation
AR	Absolute Residual $\|x_i - \hat{x}_i\|$
MRE	Magnitude of Relative Error $\frac{AR}{actual_i}$
MER	Magnitude of Error Relative $\frac{AR}{predicted_i}$
MMRE	Mean Magnitude of Relative Error $mean(MRE)$
MdMRE	Median Magnitude of Relative Error $median(MRE)$
PRED(X)	The percentage of estimates that are within X% of the actual value
BRE	Mean Balanced Relative Error $\frac{AR}{min(actual_i, predicted_i)}$
IBRE	Mean Inverted Balanced Relative Error $\frac{AR}{max(actual_i, predicted_i)}$

Table 4: The explanations of symbols that are used in this paper. Symbols that are related to each other are grouped together.

4. RESULTS

After getting results from Scott-Knott, we have consolidated these results and generated two tables 2 and 3. First table showing cumulative ranking, i.e., we have simply added the ranks provided by Scott-Knott for the solos and ranked them. Second table is based on the frequency ranking, i.e., we calculated the frequency of each solo ranked 1 by Scott-Knott.

The Table 2 shows the rank of our 48 solos, based on the cumulative ranking by Scott-Knott among various datasets / error measures. log + Cart (no) performed better in this category, as it has a total score of 138, which is the lowest

Table 5: 10 Datasets used in this Study

Dataset	Features	Size	Description
Kemerer	7	15	Large Business Applications
Albrecht	8	24	Projects from IBM
Maxwell	27	62	Projects from commercial banks in Finland
Cocomo81	17	63	NASA projects
Desharnais	12	81	Canadian Software Projects
China	18	499	Projects from Chinese Software Companies
Miyazaki94	9	48	Japanese Software projects developed in COBOL
Diabetes	9	768	Pima Indians Diabetes Database
Ivy-1.1	21	111	
Jedit-4.1	21	311	
		Total: 1982	

among all the solos. In this method, solo getting the lowest cumulative score should be ranked better. It is because solo getting lower score means that the solo has performed better in some dataset/error measures, i.e. got better rank. Learner wise, we can see that Cart (no) has better results when comparing with all other learners.

The Table 3 shows the rank of our 48 solos, based on the frequency of solos being ranked by the Scott-Knott as rank 1. log + Cart (no) performed better and we can see from the table that it was ranked as rank 1 by Scott-Knott 42 times in the process. none + Cart (no) and norm + Cart (no) also perfomed better and ranked as rank 1 by Scott-Knott 42 times in the process. In this method, solo getting highest score is the better solo because we are calculating the consistency here i.e., solo performing better for many datasets/error measures. We can see that Cart (no) is performing better in this method as well.

5. DISCUSSION

We performed the comparision between the two ranking methods to understand, if there is trend existing between the two methods. We could note that there were few solos that were performing consistently in both the ranking methods. For example, you can notice in figure 2, that solo Log + Cart (no) was ranked first in both the methodology, which meant not only did it have multiple rank 1 in Scott-Knott but also never performed poorly in any case. We can also notice that width3bin + PLSR performed well in terms of frequency ranking, it was not consistent across all the error measures/datasets. The high cumulative ranking proves, we cannot always trust frequency rankings. On a similar note, width5bin + Cart (no) performed poorly in frequency ranking, but it has better cumulative ranking. This might be because, it was never the best solo (rank 1) but it consistently must have performed well across all datasets/error measures. This shows, we cannot trust cumulative ranking alone as well. The combination of both the methodologies would be a better choice to rank the solo methods.

6. DATA DABBLING VS DATA SCIENCE

The datasets we have used can be found at the PROMISE repository. The code can be found at GitHub, which is also an open source. The process we followed, can be easily repeatable, so the work we did is not a work of a data dabbler and it a data science project.

7. CONCLUSION

We cannot say that single solo is better for all datasets. We do not have evidence to support that single solo can perform better for all datasets. We have seen some solo preforming consistently for some datasets/error measures but there is no consistency for all datasets/error measures. The ranking of solos is difficult and the combination of our proposed methods (cumulative + frequency) has to be used to rank the solos better.

APPENDIX

A. DATA

All the data used in this study are available either at *http://promisedata.org/data* or through the authors. As shown in Table 5, we use a variety of different datasets in this research. The standard **COCOMO** datasets (cocomo*, nasa*) are collected with the COCOMO approach [5]

The **Desharnais** dataset contains software projects from Canada. It is collected with function points approach.

The **Albrecht** dataset consists of projects completed at IBM in the 1970s and details are given in [2].

The **Kemerer** is a relatively small dataset with 15 instances, whose details can be found in [9].

The **Maxwell** dataset comes from the finance domain and is composed of Finnish banking software projects.

The **China** dataset includes various software projects from multiple companies developed in China.

The **Miyazaki94** dataset includes various Japanese software projects developed in COBOl.

The **Diabetes** dataset includes datas from Pima Indians Diabetes database.

The **Ivy1.1** dataset and **Jedit-4.1** dataset is provided by Tim Menzies.

B. STEPS TO REPRODUCE OUR WORK

We have kept a file *install.py* in our repo, which can be used to install all the required packages for our project. The detailed steps to run our project is as follows:

- Python (>=2.6 or >=3.3), NumPy (>=1.6.1), and SciPy (>=0.9) are must for our project to run.
- Install Scikit-Learn using pip install -U scikit-learn or conda install scikit-learn.
- *config.py* can be used to configure datasets, solos, cross validation values and to decide whether cross validation should be done or not.

- After configuring the required values, run *tableReader.py* to start comparing the solos.
- A folder *TEMP* will be created in a working directory, and all the reports are stored in this folder.
- We are using Scott-Knott to compare the report, but users can perform their choice of operations with the generated reports.

C. REFERENCES

[1] E. Acuna and C. Rodriguez. The treatment of missing values and its effect on classifier accuracy. In *Classification, clustering, and data mining applications*, pages 639–647. Springer, 2004.

[2] A. J. Albrecht and J. E. Gaffney. Software function, source lines of code, and development effort prediction: a software science validation. *IEEE transactions on software engineering*, (6):639–648, 1983.

[3] E. Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[4] L. L. Bhering, C. D. Cruz, E. S. De Vasconcelos, A. Ferreira, and M. de Resende. Alternative methodology for scott-knott test. *CROP BREEDING AND APPLIED TECHNOLOGY*, 8(1):9, 2008.

[5] B. W. Boehm et al. *Software engineering economics*, volume 197. Prentice-hall Englewood Cliffs (NJ), 1981.

[6] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

[7] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on software engineering*, 33(1):33–53, 2007.

[8] G. Kadoda, M. Cartwright, and M. Shepperd. On configuring a case-based reasoning software project prediction system. In *UK CBR Workshop, Cambridge, UK*, pages 1–10. Citeseer, 2000.

[9] C. F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, 1987.

[10] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE Transactions on Software Engineering*, 38(2):425–438, 2012.

[11] E. Kocaguneli, T. Menzies, and J. W. Keung. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering*, 38(6):1403–1416, 2012.

[12] R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995.

[13] Y.-F. Li, M. Xie, and T. N. Goh. A study of project selection and feature weighting for analogy based software cost estimation. *Journal of Systems and Software*, 82(2):241–252, 2009.

[14] E. Mendes, I. 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.

[15] K. Pillai and V. S. Nair. A model for software development effort and cost estimation. *IEEE Transactions on Software Engineering*, 23(8):485–497, 1997.

[16] G. Seni and J. F. Elder. Ensemble methods in data mining: improving accuracy through combining predictions. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 2(1):1–126, 2010.

[17] M. Shepperd and M. Cartwright. Predicting with sparse data. *IEEE Transactions on Software Engineering*, 27(11):987–998, 2001.

[18] M. Shepperd and G. Kadoda. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, 27(11):1014–1022, 2001.

[19] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on software engineering*, 23(11):736–743, 1997.

[20] H. Zhao and S. Ram. Constrained cascade generalization of decision trees. *IEEE Transactions on Knowledge and Data Engineering*, 16(6):727–739, 2004.