Autom Softw Eng
DOI 10.1007/s10515-012-0108-5

Finding conclusion stability for selecting the best effort predictor in software effort estimation

Jacky Keung · Ekrem Kocaguneli · Tim Menzies

Received: 2 January 2011 / Accepted: 25 April 2012

Abstract ***Background:*** *Conclusion Instability* in software effort estimation (SEE) refers to the inconsistent results produced by a diversity of predictors using different datasets. This is largely due to the "ranking instability" problem, which is highly related to the evaluation criteria and the subset of the data being used.

Aim: To determine stable rankings of different predictors.

Method: 90 predictors are used with 20 datasets and evaluated using 7 performance measures, whose results are subject to Wilcoxon rank test (95 %). These results are called the "*aggregate results*". The aggregate results are challenged by a sanity check, which focuses on a single error measure (MRE) and uses a newly developed evaluation algorithm called *CLUSTER*. These results are called the "*specific results*."

Results: Aggregate results show that: (1) It is now possible to draw stable conclusions about the relative performance of SEE predictors; (2) Regression trees or analogy-based methods are the best performers. The aggregate results are also confirmed by the specific results of the sanity check.

Conclusion: This study offers means to address the conclusion instability issue in SEE, which is an important finding for empirical software engineering.

Keywords Effort estimation · Data mining · Stability · Linear regression · Regression trees · Neural nets · Analogy · MMRE · Evaluation criteria

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

E. Kocaguneli (✉) · T. Menzies
Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26505, USA
e-mail: ekocagun@mix.wvu.edu

T. Menzies
e-mail: tim@menzies.us

Published online: 12 May 2012

1 Introduction

Being able to choose the most appropriate software development effort predictor for the local software projects remains elusive for many project managers and researchers. For decades, researchers have been seeking for the "best" software effort predictor. At the time of writing, there is no such a commonly agreed "best" predictor found, which provides consistently the most accurate estimate. The usual conclusion from studies is that effort estimation suffers from a *ranking instability* syndrome; i.e. different researchers offer conflicting rankings as to what is "best" (Shepperd and Kadoda 2001; Myrtveit et al. 2005). It seems, given different historical datasets, different sets of best effort predictors exist under various different situations.

This is an open and imminent issue as accurate effort estimation is crucial to Software Engineering, and is known as a major challenge for many software project managers. Both overestimating and underestimating would result in unfavorable impacts to the business competitiveness and project resource planning. Conventionally, the single most familiar effort predictor may be used for different situations, however this approach may not produce the best effort estimates for different projects.

Being able to compare and determine the best effort predictor for different scenarios is critically important to the relevance of the estimates to the target problem. 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 demonstrated in this study, learner performance is greatly influenced by the data preprocessing and the datasets being used to evaluate the learner. A combination of a preprocessor and a learner *forms* a complete effort estimation predictor; e.g. the data normalization technique as a preprocessor with linear regression as the learner.

Ranking stability in software effort estimation is of the primary research focus. Being able to correctly classify the characteristics of each method allows the most suitable predictors to be used in the estimation process. This study is not at an early stage, it is based on the success of a previous study described in Menzies et al. (2010), where a large number of predictors were applied on COCOMO datasets, and they were able to derive precise and stable ranking of all the predictors under changing parameters in the random number seeds, different evaluation criteria and subsets of the data used.

The hypothesis in our study is that if we are able to derive a stable ranking conclusion using simulated data, similar behavior should be observed when applying real heterogeneous datasets from public domains, Note that data sets from public domains come from different sources with various differences in project characteristics and evaluation criteria. The main contribution is that this comprehensive study presents a method, which can be used to determine the best effort predictors to use at different situations.

Method combinations can produce vastly different results, in all, this study applies 90 predictors (9 preprocessors $\times$ 10 learners) to 20 datasets and measure their performance using seven performance criteria subject to a statistical check via Wilcoxon non-parametric statistical test. The statistical test is used to generate the so called

win-tie-loss statistics. One result of exploring such a large space of data and algorithms is that we are able to report stable conclusions. We will refer to our results coming from the *aggregate* of 90 predictors, 20 data sets and 7 error measures as the "*aggregate results*." The aggregate results are validated through focusing on a specific case of a single error measure and a statistical method other than win-tie-loss statistics. For the error measure we chose to use magnitude of relative error (MRE) as it is one of the mostly used performance measures in software effort estimation. As for the statistical method, we developed and applied an algorithm that is called the "*CLUSTER*," whose details are given in Sect. 4.1. *CLUSTER* first sorts 90 predictors from best to worst (according to median MRE, i.e. MdMRE) and allows each data set to group 90 predictors into a number of clusters. The methods in every cluster are statistically the same according to Kruskal-Wallis statistical test (an ANOVA alternative) and the neighboring methods in consecutive clusters (i.e. the methods where two cluster became disjoint) are statistically different from one another according to Wilcoxon statistical test. So, *CLUSTER* uses:

- Wilcoxon to compare 2 learners to see whether or not the second method should start a new cluster;
- and Kruskal-Wallis to compare n-many learners to see if the methods within a single cluster would still be statistically the same after the addition of a new learner.

This sanity check showed us that:

- Some data sets (6 out of 20) are non-diverse, i.e. for these data sets the grouping of 90 methods into clusters do not create diversity (measured with Gini index).
- It is unnecessary to seek for a sanity check in non-diverse data sets (as most of the learners are grouped into a big cluster), which gives $20 - 6 = 14$ data sets for the sanity check.
- It is possible to have data sets, where the aggregate results and the specific results do not completely agree on which group of methods is the best.
- However, such data sets are very rare (1 out of 14), i.e. aggregate results hold for most of the data sets (13 out of 14) in a specific setting.

This paper is structured as follows. Section 2 discusses effort estimation and the prior reports on *conclusion instability*. The experimental settings of this research are given in Sect. 3. Section 4 firstly summarizes the aggregate results, which show that if we extend the experiments to a broader set of methods and project data we are able to: (1) discover stable conclusions and (2) list the best (and the worst) effort predictors. In Sect. 4.1 we conduct a sanity check on the aggregate results. Then we provide the discussion of our findings and list the threats to validity in Sect. 5. Section 6 concludes the paper.

2 Background

2.1 Comparison of multiple software effort estimation methods

With the availability of different SEE methods, it is becoming a more non-trivial task to select the most appropriate modeling methods for a particular software development situation. Despite decades of research, there is still no consensus on which effort

predictors are better or worse than others. Researchers have expressed concerns and even doubt that such a ranking of predictors can ever be generated.

For example, Shepperd and Kododa (2001) 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. A number of conflicting results exist in the literature as to which method provides superior prediction accuracy, and possible explanations are offered including the misuse of an evaluation criteria such as MMRE or a malformed dataset being used etc. All of these can have a strong influence on the relative effectiveness of different prediction models. The conclusion of Shepperd et al.'s study based on simulation is that it is generally *infeasible* to determine which prediction technique is the "best":

– *None* of these existing predictors were consistently the "best";
– The accuracy of an estimate depends on the dataset characteristic and a suitable prediction model for the given dataset.

2.2 Ranking instability

More recent results suggest that it is appropriate to revisit the ranking instability hypothesis. Menzies et al. (2010) applied 158 predictors to various subsets of two COCOMO datasets. In a result consistent with Shepperd and Kododa, they found the precise ranking of the 158 predictors *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.

In addition, there are 4 methods consistently outperformed the other 154 across all datasets, across 5 different random number seeds, and across three different evaluation criteria, making the result *stable*.

The hypothesis in here is that if we are able to derive a stable conclusion using simulated data across 5 different random number seeds as source of changes, and across 3 different evaluation criteria, similar behavior should be observed when applying heterogenous datasets from public domains where they are from different sources with various differences in project characteristics. Figure 1 lists 20 real world software development project datasets which have become available at the PROMISE repository of reusable SE data.[1] This enables a more in-depth understanding of a more stable conclusion using real SE project data instead of simulated data (as reported by Shepperd and Kadoda 2001) or to study merely two datasets (as reported by Menzies et al. 2010). A more extensive empirical analysis can be carried out.

With 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 motivation is to question the stability issue; accordingly, we propose a methodology for evaluating the stability (see methodology of Fig. 6).

[1] http://promisedata.org/data.

Dataset	Features	Size	Description	Units
cocomo81	17	63	NASA projects	months
cocomo81e	17	28	Cocomo81 embedded projects	months
cocomo81o	17	24	Cocomo81 organic projects	months
cocomo81s	17	11	Cocomo81 semi-detached projects	months
nasa93	17	93	NASA projects	months
nasa93_center_1	17	12	Nasa93 projects from center 1	months
nasa93_center_2	17	37	Nasa93 projects from center 2	months
nasa93_center_5	17	40	Nasa93 projects from center 5	months
desharnais	12	81	Canadian software projects	hours
desharnaisL1	11	46	Projects in Desharnais that are developed with Language1	hours
desharnaisL2	11	25	Projects in Desharnais that are developed with Language2	hours
desharnaisL3	11	10	Projects in Desharnais that are developed with Language3	hours
sdr	22	24	Turkish software projects	months
albrecht	7	24	Projects from IBM	months
finnish	8	38	Software projects developed in Finland	hours
kemerer	7	15	Large business applications	months
maxwell	27	62	Projects from commercial banks in Finland	hours
miyazaki94	8	48	Japanese software projects developed in COBOL	months
telecom	3	18	Maintenance projects for telecom companies	months
china	18	499	Projects from Chines software companies	hours
		Total: 1198		

Fig. 1 The 1198 projects used in this study come from 20 data sets. Indentation in column one denotes a dataset that is a subset of another dataset. For notes on these datasets, see the Appendix

2.3 Estimation methods for software development projects

2.3.1 Algorithmic methods

There are different algorithmic effort predictors introduced over the past 15 years. For instance, in the class of instance-based algorithms, Fig. 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. Refer to Sect. 3.3 for further information. The combination of the instance and non-instance-based methods can create even more algorithms. For example, once an instance-based method finds its nearest neighbors, those neighboring items can be used for adaptation to the problem under investigation using regression or neural nets (Li et al. 2009).

2.3.2 Non-algorithmic methods

An alternative popular approach to algorithmic approaches (e.g. the instance-based methods of Fig. 2) is to utilize the best knowledge of an experienced expert. Expert based estimation (Jørgensen 2004) is a human intensive approach that is most commonly adopted in practice. Estimates are usually produced by domain experts 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 (e.g. when there is a lack of historical data).

Jorgensen (2005) provides guidelines for producing realistic software development effort estimates derived from industrial experience and empirical studies. An interesting finding was that the *combined estimation* method in expert based estimation offers the most robust and accurate estimation method, as combining estimates

Instance-based learners draw conclusions from instances *near* the test instance. Mendes et al. (2003) discuss various *near*-ness measures.

M_1 : A simple Euclidean measure;
M_2 : A "maximum distance" measure that 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 (Mendes et al. 2003);

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

N_1 : Using in an "*as is*" manner;
N_2 : Using outlier removal (Keung et al. 2008);
N_3 : *Prototype generation*; i.e. replace the data set with a smaller set of most representative examples (Chang 1974).

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 (Keung et al. 2008; Li et al. 2009; Hall and Holmes 2003). The pre-processors may require *discretization*.

Discretization breaks up continuous ranges at points $b_1, b_2, \ldots$, each containing counts of $c_1, c_2, \ldots$ of numbers (Gama and Pinto 2006). 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 (Fayyad and Irani 1993);
D_4 : PKID (Yang and Webb 2002);
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. (1998) and Walkerden & Jeffery (1999);
K_2 : $k = 2$ is used by Kirsopp & Shepperd (2002)
K_3 : $k = 1, 2, 3$ are used by Mendes el al. (2003)
K_4 : Li et al. use $k = 5$ (Li et al. 2009);
K_5 : Baker tuned k to a particular training set using an experimental method (Baker 2007).

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 $k = 1$ 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

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 still many ad-hoc methods for expert based estimation. Shepperd et al. (1996) do not consider expert based estimation

an empirical method because the means of deriving an estimate are not explicit and therefore not repeatable, nor are they 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.

3 Experimental design

In our experiments, numerous performance measures were collected after various *predictors* (combinations of preprocessors and learners) were applied to the data of Fig. 1. This section describes those performance measures, preprocessors, and learners.

Since it is impractical to explore (say) the thousands of options described in Fig. 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 (2001). Nevertheless, it is important to note that our conclusions come from the predictors, performance criteria and datasets used in this study. Further work may be required to confirm our findings on other predictors, performance criteria, datasets.

The following sections will provide details of the experiment on 20 datasets:

- 3.1. Evaluation Criteria
- 3.2. Pre-Processors
- 3.3. Predictors/Learners
- 3.4. Method/Procedure

3.1 Performance evaluation criteria

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

$$AR_i = |x_i - \hat{x}_i| \tag{1}$$

(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 predictor from a number of competing software prediction models (Shepperd and Schofield 1997; Foss et al. 2003). MRE measures the error ratio between the actual effort and the predicted effort and can be expressed as the following equation:

$$MRE_i = \frac{|x_i - \hat{x}_i|}{x_i} = \frac{AR_i}{x_i} \tag{2}$$

A related measure is MER (Magnitude of Error Relative to the estimate (Foss et al. 2003)):

$$MER_i = \frac{|x_i - \hat{x}_i|}{\hat{x}_i} = \frac{AR_i}{\hat{x}_i} \tag{3}$$

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

$$MMRE = mean(all\ MRE_i) \tag{4}$$

$$MdMRE = median(all\ MRE_i) \tag{5}$$

A common alternative to MMRE is PRED(25), and it is defined as the percentage of successful predictions falling within 25 % of the actual values, and can be expressed as follows, where N is the dataset size:

$$PRED(25) = \frac{100}{N} \sum_{i=1}^{N} \begin{cases} 1 & \text{if } MRE_i \leq \frac{25}{100} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

For example, PRED(25) = 50 % implies that half of the estimates are failing within 25 % of the actual values (Shepperd and Schofield 1997).

There are other performance measures including Mean Balanced Relative Error (MBRE) and the Mean Inverted Balanced Relative Error (MIBRE) studied by Foss et al. (2003):

$$MBRE_i = \frac{|\hat{x_i} - x_i|}{\min(\hat{x_i}, x_i)} \tag{7}$$

$$MIBRE_i = \frac{|\hat{x_i} - x_i|}{\max(\hat{x_i}, x_i)} \tag{8}$$

3.2 Pre-processors

In this study, we use 10 pre-processors for investigation:

- x3 *simple preprocessors*: **none**, **norm**, **and log**;
- x1 *feature synthesis* methods called **PCA**;
- x2 *feature selection* methods: **SFS** (sequential forward selection) and **SWR**;
- x4 *discretization* methods: divided on 3 and 5-bins based on equal frequency and width.

None is just the simplest *option* of avoiding a pre-processor, i.e. all data values are unadjusted. With the **norm** (max-min) preprocessor, numeric values are normalized to a 0–1 interval using (9).

$$normalizedValue = \frac{(actualValue - \min(allValues))}{(\max(allValues) - \min(allValues))} \tag{9}$$

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

Principal component analysis (Alpaydin 2004), or **PCA**, is a *feature synthesis* preprocessor that converts a number of possibly correlated variables into a smaller number of uncorrelated variables called components.

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. Whenever 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 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 was designed to work 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 map 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 (10).

$$binWidth = ceiling\left(\frac{\max(allValues) - \min(allValues)}{n}\right) \tag{10}$$

– **width5bin:** Same as **width3bin** but 5 bins instead.
– **freq3bin:** Generates 3 bins of equal population size;
– **freq5bin:** Same as **freq3bin**, but *5* bins instead.

3.3 Predictors (learners)

Based on the effort estimation literature, we identified 9 commonly used learners:

– x2 *instance-based* learners: **ABE0-1NN**, **ABE0-5NN**;
– x2 *iterative dichotomizers*: **CART(yes)**,**CART(no)**;
– x1 *neural net*: **NNet**;
– x4 *regression methods*: **LReg**, **PCR**, **PLSR**, **SWR**.

Instance-based learning can be used for analogy-based estimation. A large class of ABE algorithms was described in Fig. 2. Since it is not practical to experiment with the 6000 options defined in Fig. 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 (Mendes et al. 2003; Li et al. 2009; Kadoda et al. 2000). In **ABE0-*k*NN**, features are firstly normalized to 0–1 interval, then the distance between test and train instances is measured according to Euclidean distance function, k nearest neighbors are chosen from training set and finally for finding estimated value (a.k.a adaptation procedure) the median of k nearest neighbors is calculated. We explored two different kNN:

– **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 (Breiman et al. 1984) is defined for continuous target concepts and its *splitters* strives to reduce the GINI index of the data in each split. In this study, we use two variants:

– **CART (yes):** This version prunes the generated tree using cross-validation. For each cross-validation, an internal node 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 (which yields 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 four layers: Input layer, two hidden layers and an output layer.

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. **SWR** is the stepwise regression discussed above. Whereas above, **SWR** was used to select features for other learners, here we use **SWR** as a learner (that is, the predicted value is a regression result using the features selected by the last step of **SWR**). Partial Least Squares Regression (**PLSR**) as well as Principal Components Regression (**PCR**) are algorithms that are used to model independent variables. While modeling, they both construct new independent variables as linear combinations of original ones. 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.

3.4 Experimental procedure

This study reused the experimental procedure of a recent prominent study (Li and Ruhe 2007). In the leave-one-out experiment, given N projects, 1 project at a time is selected as the test and the remaining $N - 1$ projects are used for training, so eventually we have N predictions (this procedure refers to Jack-knifing in statistics). The resulting N predictions are then used to compute our seven evaluation criteria given in Sect. 3.1.

Fig. 3 Comparing algorithms (i, j) on their performance (P_i, P_j). P stands for a particular performance measure. The "better" predicate changes according to P. For error measures like MRE, "better" means lower values. For PRED(25), "better" means higher values

```
if WILCOXON(P_i, P_j, 95) says they are the same then
   tie_i = tie_i + 1;
   tie_j = tie_j + 1;
else
   if better( median(P_i), median(P_j)) then
      win_i = win_i + 1
      loss_j = loss_j + 1
   else
      win_j = win_j + 1
      loss_i = loss_i + 1
   end if
end if
```

To compare the performance of one predictor 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 possible outliers in the given datasets. Ranked statistical tests like the Wilcoxon are also useful if it is not clear that the underlying distributions are Gaussian (Kliijnen 1997).

Using the Wilcoxon test, for each dataset, the performance measures collected from each of our 90 predictors was compared to the 89 others. This allowed us to collect *win-tie-loss* statistics using the algorithm of Fig. 3. First, we want to check if two distributions i, j are statistically different according to the Wilcoxon test (95 % confident); otherwise we increment tie_i and tie_j. If the distributions are statistically different, we update win_i, win_j and $loss_i, loss_j$, after checking which one is *better* w.r.t. current performance measure.

4 Results

After applying leave-one-out to all 20 data sets, the procedure of Fig. 3 was repeated seven times (once for MAR, MMRE, MMER, MBRE, MIBRE, MdMRE and PRED(25)). Our ninety predictors were then sorted by their total number of losses over all datasets. The resulting sort order is shown in Fig. 4. The predictor, with fewest losses (**norm/CART(yes)**) was ranked #1 and the predictor 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 predictor was $89 \times 7 \times 20 = 12{,}460$. Figure 5 sorts all 90 predictors 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 Fig. 4; e.g. the top ranked predictor of **norm/CART(yes)** lost in nearly zero percent of our experiments.

Figure 6 tests the stability of the predictors. In this plot, we check if the sort orders are changed by different performance criteria:

- In Fig. 6, we report the mean of maximum rank changes for each predictor with respect to their ordering in Fig. 4.
  - Each error measure defines its own ordering of predictors w.r.t. its *win*, *loss* or *win* − *loss* values.

rank	# of losses	pre-processor	learner	rank	# of losses	pre-processor	learner
1	148	norm	CART (yes)	46	1668	PCA	NNet
2	150	norm	CART (no)	47	1671	width3bin	ABE0-5NN
3	151	none	CART (yes)	48	1673	none	NNet
4	152	none	CART (no)	49	1682	width5bin	SWR
5	155	log	CART (yes)	50	1727	width5bin	ABE0-1NN
6	157	log	CART (no)	51	1737	none	LReg
7	226	SWR	CART (yes)	52	1776	width5bin	ABE0-5NN
8	228	SWR	CART (no)	53	1783	SFS	NNet
9	400	SFS	CART (yes)	54	1788	norm	PLSR
10	404	SFS	CART (no)	55	1809	freq5bin	ABE0-1NN
11	412	SWR	ABE0-1NN	56	1818	SWR	NNet
12	634	log	ABE0-1NN	57	1851	SWR	LReg
13	641	SWR	ABE0-5NN	58	1876	norm	LReg
14	728	SFS	ABE0-5NN	59	1941	freq3bin	ABE0-1NN
15	732	PCA	PLSR	60	1945	freq3bin	CART (yes)
16	733	SWR	PCR	61	1945	freq3bin	CART (no)
17	734	none	PLSR	62	1961	PCA	ABE0-1NN
18	740	SFS	ABE0-1NN	63	2284	width3bin	SWR
19	749	PCA	PCR	64	2284	width5bin	PLSR
20	765	none	PCR	65	2386	log	SWR
21	888	PCA	CART (yes)	66	2515	log	PCR
22	888	PCA	CART (no)	67	2665	log	PLSR
23	907	freq5bin	ABE0-5NN	68	2725	width3bin	PLSR
24	918	SWR	PLSR	69	2729	width3bin	ABE0-1NN
25	927	SFS	LReg	70	2810	width5bin	PCR
26	929	norm	ABE0-1NN	71	2853	norm	PCR
27	933	none	ABE0-1NN	72	3413	width3bin	PCR
28	994	SFS	PCR	73	3528	freq5bin	PCR
29	999	SFS	PLSR	74	3627	freq5bin	SWR
30	1030	freq5bin	CART (yes)	75	3647	width3bin	LReg
31	1030	freq5bin	CART (no)	76	3737	freq3bin	PCR
32	1069	width5bin	CART (yes)	77	3802	width5bin	LReg
33	1069	width5bin	CART (no)	78	3822	freq3bin	PLSR
34	1123	norm	ABE0-5NN	79	3829	freq5bin	PLSR
35	1127	PCA	SWR	80	3871	log	LReg
36	1148	none	ABE0-5NN	81	4656	freq3bin	SWR
37	1231	SWR	SWR	82	5980	freq5bin	LReg
38	1269	SFS	SWR	83	6405	width5bin	NNet
39	1284	log	ABE0-5NN	84	6411	norm	NNet
40	1375	norm	SWR	85	6414	width3bin	NNet
41	1381	none	SWR	86	6417	log	NNet
42	1440	freq3bin	ABE0-5NN	87	6420	freq3bin	NNet
43	1493	PCA	ABE0-5NN	88	6422	freq5bin	NNet
44	1532	width3bin	CART (yes)	89	7065	freq3bin	LReg
45	1532	width3bin	CART (no)	90	10252	PCA	LReg

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

- Maximum rank change is the maximum absolute difference between either of these orderings.
- Then, mean of maximum rank changes coming from 7 performance measures gives us Fig. 6.

The sort order on the x-axis of Fig. 6 was kept the same as the before. A line drawn parallel to x-axis at $y = 10$ gives predictors, whose mean rank change is less/more than 10. See in Fig. 6 that $y = 10$ line divides all predictors into 3 regions: a (from predictor 1 to 13), b (from predictor 14 to 64) and c (from predictor 65 to 90). Regions a and c show "*high-ranked*" and "*low-ranked*" predictors respectively. None of

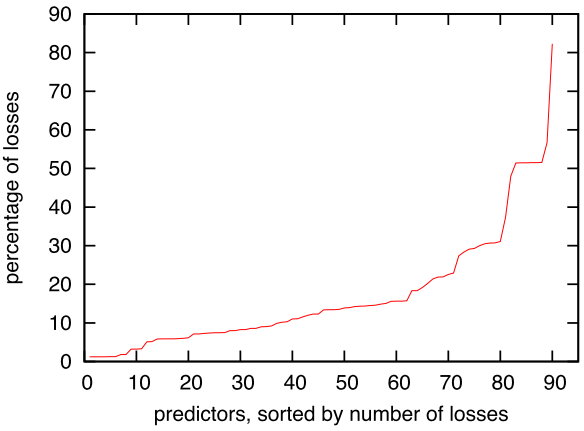

Fig. 5 The ninety predictors of Fig. 4, sorted by their percentage of maximum possible losses (so 100 % = 12,460)

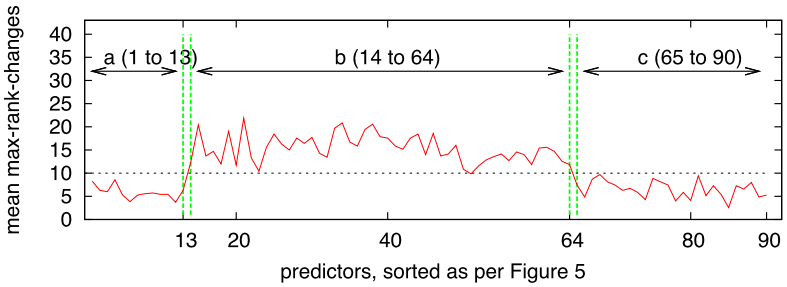

Fig. 6 Predictors and the mean of their maximum rank changes over all performance measures. Mean rank change of smaller than 10 divides 90 predictors into 3 regions. Region "a" consists of high-ranked stable predictors, whereas region "c" contains low-ranked but still stable predictors. Region "b" on the other hand shows middle-ranked and non-stable predictors

the predictors 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*" predictors are accumulated. However, all predictors 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 Fig. 5.

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 Fig. 7. In that figure, each x, y position shows the results of 623 comparisons (each predictor compared to 89 others using seven performance measures; $89 \times 7 = 623$). The y-axis of that figure shows the 90 predictors sorted in the rank order of Fig. 4. For example, the top-ranked predictor **norm/CART(yes)** appears at $y = 1$; the **log/ABE0-1NN** result appears at $y = 12$; the **log/LReg** results appear at $y = 80$; and the worst-ranked predictor **PCA/LReg** appears at $y = 90$.

In order to discuss which learners/preprocessors are "best", we divide Fig. 7 into 3 bands of Fig. 6. We reserve the lowest band from predictor 1 to 13 (containing the "best" predictors) for the region where all predictors have a mean rank change of

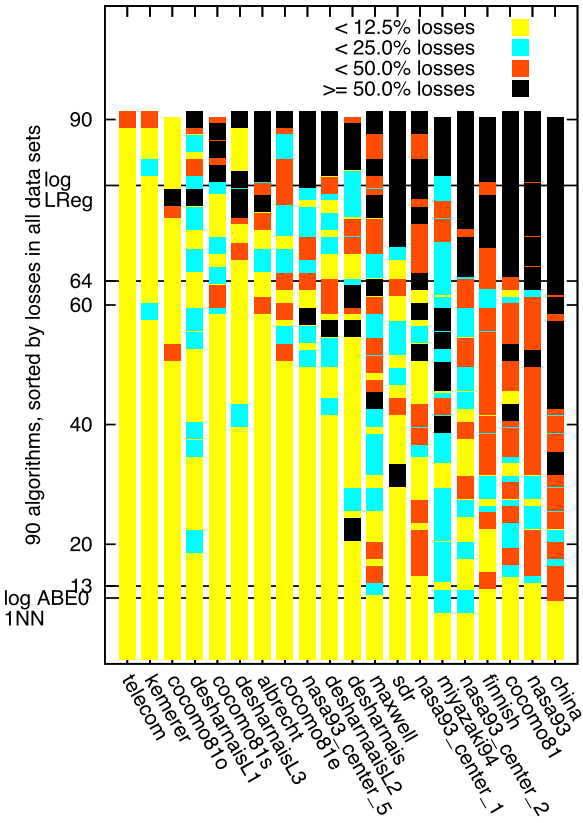

Fig. 7 Number of losses seen in 90 predictors and 20 datasets expressed as a percentage of the maximum losses possible for one predictor in one dataset (so 100 % = *89 comparisonx* × *7 error measures* = 623; 50 % = 311; 25 % = 156; 12.5 % = 78). The predictors on the *y*-axis are sorted according to Fig. 4

smaller than 10. Note that predictors in that region almost always lose less than $\frac{1}{8}$th of the time (i.e. the rows $y = 1$ to $y = 13$ that are almost completely yellow in Fig. 7). In the other bands (boundaried at $y = 14$ to $y = 64$ and $y = 65$ to $y = 90$), predictors lose much more frequently, i.e. behavior of predictors in the loss percentage graph of Fig. 7 are in agreement with the rank change graph of Fig. 6.

Figure 8 shows the spectrum of PRED(25) values across the 3 bands. As might be expected, the y-axis sort order of Fig. 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 predictors in each band.

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

– A "good" preprocessor/learner appears often in the lower bands (tendency towards band a). In Fig. 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 Fig. 9, all the discretization preprocessors (e.g. **freq3bin**) are "poor" preprocessors.
– The gray rows of Fig. 9 shows preprocessor/learner that are neither "good" nor "poor" (since they exist in all 3 bands and have high frequency counts in bands b and c); e.g. see the **log** preprocessor.

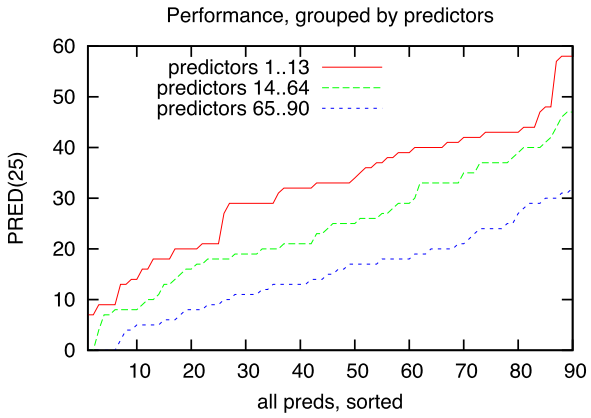

Fig. 8 Spectrum of Pred(25) across the bands

		Occurrence of learners in bands a, b, c		
		band **a**	band **b**	band **c**
	$y =$	1..13	14..64	65..90
Learners	CART (yes)	34	28	1
	CART (no)	33	28	2
	ABE0-5NN	6	55	2
	ABE0-1NN	11	44	8
	PCR	3	29	31
	PLSR	3	35	25
	LReg		22	41
	SWR		46	17
	NNet		20	43
Pre-Processors	SWR	25	37	1
	SFS	14	49	
	none	14	48	1
	log	20	17	26
	norm	14	33	16
	PCA	4	49	10
	freq5bin		28	35
	width3bin		31	32
	width5bin		42	21
	freq3bin		23	40

Fig. 9 Frequency counts over 7 error measures for preprocessor and learners in the three bands of Fig. 6

4.1 Sanity check

Our purpose in the sanity check is two-fold:

- To ensure our observations from the aggregate of loss values over 7 error measures and 20 data set would hold for a specific case due to a single error measure;
- Subjecting a specific error measure to a statistical procedure other than *win-tie-loss* statistics.

We chose to focus on the specific error measure of MRE. As for the proposed statistical assessment, we devised an algorithm called *CLUSTER*, which makes use of 2 statistical tests in combination: Wilcoxon and Kruskal-Wallis (an ANOVA alternative for the cases where ANOVA's normality assumptions may be invalid). For each data

set, *CLUSTER* groups 90 predictors into *"c"* clusters. The predictor(s) grouped in each one of the *"c"* clusters have statistically the same MRE values with one another according to Kruskal-Wallis. The neighboring predictors in two consecutive clusters have statistically different MRE values from one another according to Wilcoxon. The detailed steps of CLUSTER are as follows:

1. Take a single data set D
2. Sort 90 predictors according to their MdMRE values for D in ascending order (i.e. best predictor appears at #1).
3. Set $i = 1$ and $j = 1$
4. Place predictor #i into group #j.
5. Compare MRE values of predictor #i with #$i + 1$ w.r.t. Wilcoxon
6. Compare if MRE values of all the predictor(s) in group #j and MRE values of predictor #$i + 1$ w.r.t. Kruskal-Wallis.
7. If both Wilcoxon and Kruskal-Wallis indicate MRE values are statistically the same; then set $i = i + 1$ and go to Step 4; else set $i = i + 1$ and $j = j + 1$ and go to Step 4. Do this until all the 90 predictors are exhausted.

Note that the procedure of *CLUSTER* enables each data set to define its own groups of predictors (clusters), where MRE values of the predictors within each group are statistically the same. The number of groups formed for each data set through the *CLUSTER* as well as the number of predictors appearing in each group are given in Fig. 10. Except the china data set, all the data sets have less than 6 groups. For the reasons of space, we summed the number of predictors appearing in groups #7 to #16 of china under the *Rest* column. The number distribution of predictors into groups #7 to #16 are: 1, 9, 4, 8, 7, 7, 6, 1, 1, 6 (respectively).

Dataset	# Groups	G1	G2	G3	G4	G5	G6	Rest	Gini	Diversity
nasa93	6	20	27	1	10	25	7	-	0.76	Diverse
cocomo81	4	37	12	24	17	-	-	-	0.71	
miyazaki94	4	22	42	13	13	-	-	-	0.68	
nasa93center2	3	31	36	23	-	-	-	-	0.66	
maxwell	4	14	31	41	4	-	-	-	0.65	
china	16	2	10	7	16	2	3	50	0.60	
sdr	3	42	44	4	-	-	-	-	0.54	Semi-Diverse
desharnais	3	53	30	7	-	-	-	-	0.54	
finnish	3	23	57	10	-	-	-	-	0.52	
cocomo81e	2	48	42	-	-	-	-	-	0.50	
nasa93center1	3	67	16	7	-	-	-	-	0.41	
desharnaisL3	4	2	67	18	3	-	-	-	0.40	
nasa93center5	2	65	25	-	-	-	-	-	0.40	
cocomo81s	2	70	20	-	-	-	-	-	0.35	
albrecht	2	76	14	-	-	-	-	-	0.26	Non-Diverse
desharnaisL1	3	80	9	1	-	-	-	-	0.20	
desharnaisL2	2	80	10	-	-	-	-	-	0.20	
cocomo81o	3	1	88	1	-	-	-	-	0.04	
telecom1	1	90	-	-	-	-	-	-	0.00	
kemerer	1	90	-	-	-	-	-	-	0.00	

Fig. 10 The number distribution of 90 predictors to the groups formed by *CLUSTER* as well as the Gini indices (Breiman 1996) calculated by that distribution. Data sets are sorted w.r.t. their Gini indices in descending order

The group counts as well as the distribution of the 90 predictors to these groups for each data set can be used as an indicator of the impurity of the data sets. One of the most commonly used measures of impurity is the Gini index (Breiman 1996), which had also been used as a splitting criterion in the classification and regression trees (Breiman et al. 1984). The formula for the Gini index of a data set D with "c" clusters is given in (11), where $|c_i|$ denotes the number of predictors within i^{th} cluster.

$$Gini(D) = 1 - \sum_{i=1}^{c} \left(\frac{|c_i|}{90} \right)^2 \tag{11}$$

Given that the data set D forms only a single cluster of 90 predictors, then its Gini index becomes $Gini(D) = 1 - (1)^2 = 0$. Such *pure* data sets with a Gini index of zero are "*non-diverse*" data sets, meaning that they are unable to help us identify predictors with high and low MRE values. telecom1 and kemerer data sets are examples to "*non-diverse*" data sets.

Note that the data sets of Fig. 10 are sorted according to their Gini indices. Starting from Gini index of 0 we grouped all the 20 data sets into 3 bins with increment of 30 in the Gini index. The resulting bins are called: Diverse (with 6 data sets), semi-diverse (with 8 data sets) and non-diverse (with 6 data sets). These bins are indicated on Fig. 10. When performing our sanity check, we will focus on diverse and semi-diverse data sets only.

Figure 11 is our sanity check on the top 13 learners. It shows the success of top 13 learners in the diverse and semi-diverse data sets. The cells of Fig. 11 show in which group (top 1, 2 and so on) each one of the top 13 learners appeared. The cells in which top 13 learners do not appear within the best group (i.e. top "*1*") are highlighted. Note that there are only two data sets for which the majority of the cells are highlighted:

Dataset	# Groups	norm CART (yes)	norm CART (no)	none CART (yes)	none CART (no)	log CART (yes)	log CART (no)	SWR CART (yes)	SWR CART (no)	SFS CART (yes)	SFS CART (no)	SWR ABE0-1NN	log ABE0-1NN	SWR ABE0-5NN
nasa93	6	1	1	1	1	1	1	1	1	1	1	1	1	1
cocomo81	4	1	1	1	1	1	1	1	1	1	1	1	1	1
miyazaki94	4	1	1	1	1	1	1	1	1	2	2	1	2	1
nasa93center2	3	1	1	1	1	1	1	1	1	1	1	1	1	2
maxwell	4	1	1	1	1	1	1	2	2	1	1	2	2	2
china	16	2	2	2	2	2	2	2	2	2	2	8	8	8
sdr	3	1	1	1	1	1	1	1	1	1	1	1	1	1
desharnais	3	1	1	1	1	1	1	1	1	1	1	1	2	1
finnish	3	1	1	1	1	1	1	1	1	2	2	1	2	1
cocomo81e	2	1	1	1	1	1	1	1	1	1	1	1	1	1
nasa93center1	3	1	1	1	1	1	1	1	1	1	1	1	1	1
desharnaisL3	4	3	3	3	3	3	3	3	3	3	3	2	2	2
nasa93center5	2	1	1	1	1	1	1	1	1	1	1	1	1	1
cocomo81s	2	1	1	1	1	1	1	1	1	1	1	1	1	1

Fig. 11 The number of groups per dataset (only diverse and semi-diverse data sets) and the group-rank of the top 13 learners. The cells where the top 13 predictors do not appear on the top group are highlighted

Dataset	# Groups	log_SWR	log_PCR	log_PLSR	width3bin_PLSR	width3bin_1NN	width5bin_PCR	norm_PCR	width3bin_PCR	freq5bin_PCR	freq5bin_SWR	width3bin_LReg	freq3bin_PCR	width5bin_LReg	freq3bin_PLSR	freq5bin_PLSR	log_LReg	freq3bin_SWR	freq5bin_LReg	width5bin_NNet	norm_NNet	width3bin_NNet	log_NNet	freq3bin_NNet	freq5bin_NNet	freq3bin_LReg	PCA_LReg
nasa93	6	5	6	6	5	4	5	5	5	5	6	5	5	5	5	5	5	5	5	5	6	5	6	6	6	5	5
cocomo81	4	4	4	4	3	1	3	3	3	4	4	4	4	3	3	4	4	4	4	4	4	4	4	4	4	3	3
miyazaki94	4	4	4	4	2	3	2	3	2	3	3	4	3	4	3	4	4	3	3	4	4	4	4	4	4	3	2
nasa93center2	3	2	3	3	2	2	3	3	3	3	3	2	3	2	3	3	2	3	2	3	3	3	3	3	3	2	3
maxwell	4	3	3	3	3	3	3	3	3	3	3	3	3	2	3	3	3	2	3	4	4	3	3	4	4	3	3
china	16	13	12	16	11	10	12	11	13	13	13	11	12	10	13	15	13	12	12	16	16	16	14	16	16	12	12
sdr	3	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	2	3	2	2	2	2	2	2	3	3
desharnais	3	2	2	2	1	2	2	2	2	2	2	2	1	2	2	2	2	2	2	3	3	3	3	3	3	2	3
finnish	3	2	2	2	2	2	2	3	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	2	3
cocomo81e	2	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2
nasa93center1	3	1	1	2	2	1	2	2	2	2	2	1	2	1	2	2	2	2	1	3	3	3	3	3	3	2	3
desharnaisL3	4	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	4	2	4	2	2	2	2	2	2	2	4
nasa93center5	2	2	2	2	2	1	1	2	1	1	2	1	1	1	2	2	2	2	2	2	2	2	2	2	2	1	2
cocomo81s	2	2	2	1	2	1	2	2	1	2	1	2	2	2	2	2	1	2	2	1	1	1	1	1	1	2	2

Fig. 12 The number of groups per dataset and the group-rank of the bottom 26 learners to these groups. The cells where the bottom 26 learners appear on the top group are highlighted

china and desharnaisL3. Thinking that china data set has 16 groups and that the top group has only 2 predictors in it, top 13 learners to be in the 2nd best group is not dramatic result either. Among the 14 data sets, there is only 1 contradictory example, where top 13 learners found through an aggregate analysis do not hold for a specific analysis.

Our sanity check on the worst performing (i.e. bottom) 26 learners is given in Fig. 12, which also only shows the diverse and semi-diverse data sets. Figure 12 shows in which group (top 1, 2 and so on) bottom 26 learners appeared. The cells where the bottom 26 learners appear within the best group (i.e. top "*1*") are highlighted. The data set that contradicts the most with our aggregate conclusions is cocomo81s, which has 11 highlighted cells. For the rest of the data sets, the bottom 26 learners, which were identified through an aggregate analysis also perform poorly in a specific analysis.

In the sanity check, we checked the performance of the top and bottom learners of the aggregate analysis in a specific scenario. In this specific scenario, we focused our attention to a single error measure (MRE) and we also used a statistical procedure other than the win-tie-loss statistics. For both top and bottom learners we saw contradictory cases, where the results of the aggregate analysis did not hold for the specific case. However, these cases were only 1 out of 14 data sets. Hence, we can conclude that our results presented in this paper through an aggregate analysis are most likely to be valid for a specific case too.

5 Discussion

5.1 Findings

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

Finding1: Observing the small amounts of fluctuation (or jitter) in Fig. 6 we can see that our results may not be considered 100 % stable, however they are sufficiently

stable to draw conclusions. We conjecture that prior reports on ranking instability could stem from using too few data sets or too few predictors.

Finding2: Figure 4 shows the preprocessor and learner combination is important, as their current rank can be changed if a different preprocessor is used in combination with the learner. For example, the top-ranked predictor 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 significantly altered by seemingly trivial details relating to data preprocessing, as it will change the dataset characteristics input to the learner. Hence, in future, researchers should explore learners *and* the preprocessors, as they are both equally important.

Finding3: Observe in Fig. 9 how **SWR**, **LReg** and **NNet** are standout learners that fall entirely into the worst two bands. Proponents of these learners need to defend their value for the purposes of effort estimation.

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 (Boehm 1981). Yet as shown in Fig. 7, w.r.t. loss values over all error measures, **log/LReg** ranks very poorly (position 80 out of a maximum of 90 predictors). 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.

Finding4: 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 learners.

Finding5: While simple regression learners like **LReg** are deprecated by this study, more intricate regression learners 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**.

Finding6: The top-ranked learner was **norm/CART(yes)**.

Finding7: Simple predictors (e.g. $k = 1$ nearest neighbor on the log of the numerics) perform nearly as well as the top-ranked predictor. Figure 13 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 predictor. Except for China dataset, the difference in PRED(25) values is either slightly negative, or positive. That is, even though the rank = 1 predictor is *relatively* "best" (measured according to our comparative Wilcoxon tests), when measured in an *absolute* sense, it is not impressively better than simpler alternatives.

Finding7 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 $effort \propto KLOC^x$). The fact that non-parametric instance predictors perform nearly as well as our best predictor 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.

Fig. 13 Using all data sets to compare the Pred(25) of **norm/CART(yes)** (rank = 1) and *log/ABE0-1NN* (rank = 12)

	norm/CART(yes)	log/ABE0-1NN	difference
kemerer	7	27	-20
desharnaisL3	20	40	-20
nasa93_center_2	43	57	-14
nasa93	29	39	-10
cocomo81s	9	18	-9
albrecht	33	42	-9
telecom1	33	39	-6
cocomo81	13	16	-3
nasa93_center_5	36	33	3
desharnaisL1	39	35	4
cocomo81o	29	21	8
desharnaisL2	48	40	8
cocomo81e	18	7	11
desharnais	43	32	11
sdr	42	29	13
miyazaki94	40	25	15
maxwell	32	15	17
finnish	61	37	24
nasa93_center_1	58	33	25
china	95	43	52

Secondly, analogy-based estimation methods are widely used (Auer et al. 2006; Walkerden and Jeffery 1999; Kirsopp et al. 2003; Shepperd et al. 1996; Li and Ruhe 2006; Li and Ruhe 2007; Li et al. 2009; Keung 2008; Keung et al. 2008; Keung and Kitchenham 2008). Finding7 says that while this approach may not be 100 % optimal in all cases, compared to our best predictor 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 predictors (like the **log/ABE0-1NN** predictor at rank = 12) than more complex predictors (like the **norm/CART** predictor 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 (Brady and Menzies 2010). Such experimentation would have been hindered if we tried to modify the more complex CART learner (particularly if we included sub-tree pruning).

Finding8: The aggregate results are "*mostly*" confirmed by the specific results in our study. This has two implications: (1) The aggregate analysis on a large space of predictors and data sets helps us derive stable conclusions that are valid for most of the cases. (2) Practitioners should be cautious that—although being rare, e.g. 1 out of 14 data sets—there are specific cases where aggregate results may not apply.

5.2 Validity

Construct validity (i.e. face validity) assures that we are measuring what we actually intended to measure (Robson 2002). Previous studies have concerned themselves with the construct validity of different performance measures for effort estimation (e.g. Foss et al. 2003). While, in theory, these performance measures have an impact on the rankings of effort estimation predictors, we have found that other factors dominate. For example, Fig. 7 showed that some of the datasets have a major impact on what could be concluded after studying a particular estimator on these datasets. We also show empirically the surprising result that our results regarding predictors are

stable across a range of performance criteria. Furthermore, for a majority of the data sets (13 out of 14) these results (aggregate results) hold in the case of a specific sanity check.

External validity is the ability to generalize results outside the specifications of that study (Milicic and Wohlin 2004). 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 (Bakir et al. 2010). Our reading of the literature is that this study uses more data, from more sources, than numerous other papers. For example, Table 4 of Kitchenham et al. (2007) 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 predictors, recalling Fig. 2, it is clear that this study has not explored the full range of effort estimation predictors. Clearly, future work is required to repeat this study using the "best of breed" found here (e.g. bands "a" and "b" of Fig. 9 as well as other predictors).

Having cast doubts on our selection of predictors, we hasten to add that this paper has focused on predictors that have been extensively studied in the literature (Shepperd and Schofield 1997) 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 of the current published literature on effort estimation.

One last point: the analysis of this paper crucially rests on the results of Fig. 7. It is therefore prudent to consider the validity of this figure. In that figure, the majority result was that most methods tied for most methods. There are two reasons why this figure might *not* be valid:

- These methods are actually different, but our statistical tests are not of sufficient power to distinguish them apart.
- The methods we are using as so weak that there is no way to tell them apart.

If these two *theoretical* problems are *actual* problems, then that would be an issue not only for this paper but for the entire field of effort estimation. Our reading the of effort estimation literature is that the methods in Fig. 7 are in widespread use throughout this field (indeed, much of Boehm's entire career is based on **log/LReg**). As to the statistical tests used in this paper, they are also quite standard. For example, in Table 3 of the extensive literature review of Kitchenham et al. (2007), those authors list the statistical methods used in papers that past all of Kitchenham et al.'s sanity checks that, otherwise, would have rejected an "reasonable" analysis. We note that the Wilcoxon method used in this paper features prominently in that table.

6 Conclusion

In this study, 10 learners and 9 data preprocessors were combined into 90 effort estimation predictors. These were applied to 20 datasets. Performance was measured

using 7 performance indicators (AR, MRE, MER, MdMRE, MMRE, PRED(25), MBIRE). Performances were compared using a Wilcoxon ranked test (95 %). This procedure can be used as a ranking stability indicator for selecting the most suitable effort predictor in SEE, which is an important stage in the estimation process. Eight findings are noteworthy:

1. Prior reports of ranking instability about effort estimation may have been overly pessimistic. 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 predictors.
2. The effectiveness of a learner used for effort estimation (e.g. regression or analogy methods) can be significantly altered by data preprocessing (e.g. logging all numbers or normalizing 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 preprocessor that normalized the numerics into the range zero to one).
7. Very simple predictors (e.g. $K = 1$ nearest neighbor on the log of the numerics) performed nearly as well as regression trees.
8. It is important to validate the stable conclusions derived from aggregate results through specific scenarios. Such a validation in this study showed that stable conclusions hold for the specific scenarios. Yet, it is still a possibility that "*rarely*" the aggregate and specific results will favor different predictors.

Lastly, we offer an hypothesis on why certain predictors were better than others. Recall from Fig. 4 that none of the top 13 ranked predictors fit single model to the training data:

– The CART regression tree learner appears at ranks 1 through 10 of Figure 6. Each branch of a regression tree defines one context in which an estimate may be different.
– Analogy-based estimation (ABE) appears at ranks 11, 12, 13. ABE builds a different model for each test instance (using the test instances k-th nearest neighbors).

Based on this observation, we conjecture that it may be a mistake to fit a single model to effort data. Software engineering is a highly idiosyncratic process where highly trained engineers produce novel solutions for rapidly changing business situations using toolkits and languages that are constantly evolving. Hence, it seems unlikely that effort models conform to a single distribution. In terms of future directions in effort estimation, we speculate that the next generation of models will explore *combinations* of multiple predictors.

Acknowledgements This research has been funded by the Qatar/West Virginia University research grant NPRP 09-12-5-2-470.

Appendix: Data Used in This Study

All the data used in this study is available either at http://promisedata.org/data or through the authors. As shown in Fig. 1, we use a variety of different data sets in this research. The standard **COCOMO** data sets (cocomo*, nasa*), which are collected with the COCOMO approach (Boehm 1981). The **desharnais** data set, which contains software projects from Canada. It is collected with function points approach. **SDR**, which contains data from projects of various software companies in Turkey. SDR is collected by Softlab, the Bogazici University Software Engineering Research Laboratory (Bakir et al. 2010). **albrecht** data set consists of projects completed in IBM in the 1970's and details are given in Albrecht and Gaffney (1983). **finnish** data set originally contains 40 projects from different companies and data were collected by a single person. The two projects with missing values are omitted here, hence we use 38 instances. More details can be found in Kitchenham and Känsälä (1993). **kemerer** is a relatively small dataset with 15 instances, whose details can be found in Kemerer (1987). **maxwell** data set comes from finance domain and is composed of Finnish banking software projects. Details of this dataset are given in Maxwell (2002). **miyazaki** data set contains projects developed in COBOL. For details see Miyazaki et al. (1994). **telecom** contains projects which are enhancements to a U.K. telecommunication product and details are provided in Shepperd and Schofield (1997). **china** dataset includes various software projects from multiple companies developed in China.

References

Albrecht, A., Gaffney, J.: Software function, source lines of code and development effort prediction: a software science validation. IEEE Trans. Softw. Eng. **9**, 639–648 (1983)

Alpaydin, E.: Introduction to Machine Learning. MIT Press, Cambridge (2004)

Auer, M., Trendowicz, A., Graser, B., Haunschmid, E., Biffl, S.: Optimal project feature weights in analogy-based cost estimation: improvement and limitations. IEEE Trans. Softw. Eng. **32**, 83–92 (2006)

Baker, D.: A hybrid approach to expert and model-based effort estimation. Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University (2007). Available from https://eidr.wvu.edu/etd/documentdata.eTD?documentid=5443

Bakir, A., Turhan, B., Bener, A.B.: A new perspective on data homogeneity in software cost estimation: a study in the embedded systems domain. Softw. Qual. Control **18**, 57–80 (2010)

Boehm, B.W.: Software Engineering Economics. Prentice Hall PTR, Upper Saddle River (1981)

Brady, A., Menzies, T.: Case-based reasoning vs parametric models for software quality optimization. In: International Conference on Predictive Models in Software Engineering PROMISE'10, Sept. IEEE, New York (2010)

Breiman, L.: Technical note: some properties of splitting criteria. Mach. Learn. **24**(41–47), 10 (1996) doi:1023/A:1018094028462

Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Wadsworth and Brooks, Monterey (1984)

Chang, C.-L.: Finding prototypes for nearest neighbor classifiers. IEEE Trans. Comput. **C-23**(11), 1179–1184 (1974)

Fayyad, U.M., Irani, I.H.: Multi-interval discretization of continuous-valued attributes for classification learning. In: Proceedings of the International Joint Conference on Uncertainty in AI, pp. 1022–1027 (1993)

Foss, T., Stensrud, E., Kitchenham, B., Myrtveit, I.: A simulation study of the model evaluation criterion mmre. IEEE Trans. Softw. Eng. **29**(11), 985–995 (2003)

Gama, J., Pinto, C.: Discretization from data streams: applications to histograms and data mining. In: SAC '06: Proceedings of the 2006 ACM Symposium on Applied Computing, pp. 662–667. ACM Press, New York (2006). Available from http://www.liacc.up.pt/~jgama/IWKDDS/Papers/p6.pdf

Hall, M., Holmes, G.: Benchmarking attribute selection techniques for discrete class data mining. IEEE Trans. Knowl. Data Eng. **15**(6), 1437–1447 (2003)

Jørgensen, M.: A review of studies on expert estimation of software development effort. J. Syst. Softw. **70**(1–2), 37–60 (2004)

Jorgensen, M.: Practical guidelines for expert-judgment-based software effort estimation. IEEE Softw. **22**(3), 57–63 (2005)

Kadoda, G., Cartwright, M., Shepperd, M.: On configuring a case-based reasoning software project prediction system. In: UK CBR Workshop, Cambridge, UK, pp. 1–10 (2000)

Kemerer, C.: An empirical validation of software cost estimation models. Commun. ACM **30**(5), 416–429 (1987)

Keung, J.: Empirical evaluation of analogy-x for software cost estimation. In: ESEM '08: Proceedings of the Second International Symposium on Empirical Software Engineering and Measurement, pp. 294–296. ACM, New York (2008)

Keung, J., Kitchenham, B.: Experiments with analogy-x for software cost estimation. In: ASWEC '08: Proceedings of the 19th Australian Conference on Software Engineering, pp. 229–238. IEEE Computer Society, Washington (2008)

Keung, J.W., Kitchenham, B.A., Jeffery, D.R.: Analogy-x: providing statistical inference to analogy-based software cost estimation. IEEE Trans. Softw. Eng. **34**(4), 471–484 (2008)

Kirsopp, C., Shepperd, M., Premrag, R.: Case and feature subset selection in case-based software project effort prediction. In: Research and Development in Intelligent Systems XIX: Proceedings of ES2002, the Twenty-Second SGAI International Conference on Knowledge Based Systems and Applied Artificial Intelligence, p. 61 (2003)

Kirsopp, C., Shepperd, M.J.: Making inferences with small numbers of training sets. IEEE Softw. **149**(5), 123–130 (2002)

Kitchenham, B., Känsälä, K.: Inter-item correlations among function points. In: ICSE'93:Proceedings of the 15th International Conference on Software Engineering, ICSE '93, pp. 477–480. IEEE Computer Society Press, Los Alamitos (1993)

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

Kliijnen, J.: Sensitivity analysis and related analyses: a survey of statistical techniques. J. Stat. Comput. Simul. **57**(1–4), 111–142 (1997)

Li, J., Ruhe, G.: A comparative study of attribute weighting heuristics for effort estimation by analogy. In: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering, p. 74 (2006)

Li, J., Ruhe, G.: Decision support analysis for software effort estimation by analogy. In: International Conference on Predictive Models in Software Engineering PROMISE'07, May (2007)

Li, Y., Xie, M., Goh, T.: A study of project selection and feature weighting for analogy based software cost estimation. J. Syst. Softw. **82**, 241–252 (2009)

Lipowezky, U.: Selection of the optimal prototype subset for 1-nn classification. Pattern Recognit. Lett. **19**, 907–918 (1998)

Maxwell, K.D.: Applied Statistics for Software Managers. Prentice Hall, PTR, Upper Saddle River (2002)

Mendes, E., Watson, I.D., Triggs, C., Mosley, N., Counsell, S.: A comparative study of cost estimation models for web hypermedia applications. Empir. Softw. Eng. **8**(2), 163–196 (2003)

Menzies, T., Jalali, O., Hihn, J., Baker, D., Lum, K.: Stable rankings for different effort models. Autom. Softw. Eng. **17**, 409–437 (2010)

Milicic, D., Wohlin, C.: Distribution patterns of effort estimations. In: EUROMICRO, pp. 422–429 (2004)

Miyazaki, Y., Terakado, M., Ozaki, K., Nozaki, H.: Robust regression for developing software estimation models. J. Syst. Softw. **27**(1), 3–16 (1994)

Myrtveit, I., Stensrud, E., Shepperd, M.: Reliability and validity in comparative studies of software prediction models. IEEE Trans. Softw. Eng. **31**, 380–391 (2005)

Robson, C.: Real World Research: A Resource for Social Scientists and Practitioner-Researchers. Blackwell Publisher Ltd, Oxford (2002)

Shepperd, M., Kadoda, G.: Comparing software prediction techniques using simulation. IEEE Trans. Softw. Eng. **27**(11), 1014–1022 (2001)

Shepperd, M., Schofield, C.: Estimating software project effort using analogies. IEEE Trans. Softw. Eng. **23**(11), 736–743 (1997)

Shepperd, M., Schofield, C., Kitchenham, B.: Effort estimation using analogy. In: Proceedings of the 18th International Conference on Software Engineering, pp. 170–178 (1996)

Walkerden, F., Jeffery, R.: An empirical study of analogy-based software effort estimation. Empir. Softw. Eng. **4**(2), 135–158 (1999)

Yang, Y., Webb, G.I.: A comparative study of discretization methods for naive-Bayes classifiers. In: Proceedings of PKAW 2002: The 2002 Pacific Rim Knowledge Acquisition Workshop, pp. 159–173 (2002)