# Evaluating Effort Estimators for Software Engineering

Vincent Rogers, William Sica
CS, WVU, Morgantown, USA
vrogers@mix.wvu.edu,
wsica@mix.wvu.edu

Tim Menzies
CS & EE, WVU, Morgantown, USA
tim@menzies.us

## ABSTRACT

Estimating the effort required for software projects is a difficult task in software project planning. There are multiple algorithms to estimate effort from previous projects, but not a standard criteria upon which to evaluate the algorithms accuracy. This paper tests multiple effort estimation algorithms using multiple error measures from the literature, to see if different error measures provide a stable ranking of algorithms. The results indicate instability in rankings between different error method, suggesting a need for more robust ways of algorithm comparison for effort estimation.

## Categories and Subject Descriptors

[**Software Engineering**]: Software Metrics; [**Data Mining**]: Miscellaneous

## General Terms

Software effort Estimation, Analogy, MMRE, Evaluation Criteria

## Keywords

Software Effort Estimation, MMRE, Evaluation Criteria

## 1. INTRODUCTION

Effort estimation is a crucial part of software project planning. Improper estimates can lead to consequences ranging from delays to project cancellation. In a survey of the field, it was found that most projects (60-80%) encountered effort overruns, schedule overruns, or both. [7]

In order to address the need for better software project planning methods, the field of search based software engineering has been developed. [3] The field suggests stochastic search processes can be applied to existing algorithms in order to improve performance of algorithmic estimation techniques. Search based software engineering has grown since its inception, but there are still many open research questions. Harman suggests that a promising area of future work is finding a stopping criteria that relies on comparison between the similarity of proposed solutions. [2]

This requires an effective way of comparing the solutions two algorithms generate, to assess their similarity. There are numerous ways to approach this, as there is no standard performance measure for evaluating effort estimator performance. This has been a criticism against the field of empirical software engineering [6] such that better performance measures could also have an impact in the larger software engineering community.

This paper seeks to address both concerns, by analyzing different performance measures impact on algorithm rankings. This is done using a combination of algorithms approach (COMBA), in which a data preprocessor and learner and combined. This creates a large number of test algorithms from a smaller number of preprocessors and learners, creating a wide range of example solutions for testing ranking methods. This approach has been applied to effort estimation previously, in which it was found that certain datasets distinguish between algorithm performance more strongly and thus more useful for optimizing effort estimator performance. [5] This paper will additionally try to reproduce and verify the results of the previous study.

The paper is laid out as follows:
Section 2 will cover previous works regarding this subject.
Section 3 will describe the experimental procedure used.
Section 4 provides the results of the experiment.
Section 5 concludes and suggests future works using a combination of algorithms approach.

## 2. RELATED WORK

For this section, the main focus is on the conclusion instability problem. Though this project does present many results, there has yet been sturdy evidence that it can give a definitive answer as to which data miner/learner is better than all the rest or maybe even tell what kind of situations. Our secondary source is the opposite of algorithmic learners but more so on the knowledge from a human exert in a field of study, and how the combination of the two yields a higher accuracy then either on an individual basis.

### 2.1 Sturdy Conclusion Predicament

This program attempts to determine which prediction model would be considered the best among all the others it's compared against. Finding the right way to use data retrieved from a prediction model to acquire such a certain decision is a tough decision, and there are many ways to try and go about doing such a feat. Shepperd and Kododa [8] did such a study, stating that they think that as the dependence on technology is ever increasing the data sets used for data mining will, of course, also become bigger and thus more complex. Their study only used the prediction systems: regression,

*ICSE* 2011 Waikiki, Honolulu, Hawaii USA

rule induction, nearest neighbor, and neural nets, but attempted on synthetically enlarged data sets. From those they hoped to ascertain the relationship of answer accuracy, the choice between prediction models, and characteristics of data sets.

They used various error measures as well, such as MMRE, and studied the information given to try and collect correlations. From their studies they concluded that they couldn't accurately pick a "best" data miner, but they did see a dependency of an estimate's accuracy on the characteristic of a data set and data miner being used.

Currently, a data set repository, called PROMISE, offers a large, growing selection of actual data sets of varying size, complexity, and subject. So, our project doesn't have to worry about artificially enlarging data sets.

## 2.2 Non-Algorithmic Method

Acquiring an estimate, used to be done only by humans with a vast data base of knowledge and experience in a certain field. But, now estimation over certain data sets is becoming increasingly hard AND that computers have been becoming increasingly better at "learning". A person by the name of Jorgensen [4] did a study only how well the two prediction methods work together. It was found that it was more likely to influence the accuracy of the estimate when the two were used together.

The problem with that kind of human knowledge is that it's not easily passed on, and it's hard to explain how some conclusions are reached. Though, the same problem of sometimes not being to explain how an answer was conjured is also a flaw of algorithmic estimation solutions.

# 3. EXPERIMENT DESIGN

Numerous algorithmic methods exist to make estimations on effort. While this paper can not comprehensively assess all of them, it assesses a subsection of algorithms using a combination of algorithms approach. In this way, different data preprocessors are combined with different learner algorithms, so that a larger number of algorithms are generated with the addition of a new preprocessor or learner. The combination done involves sending the data to the preprocessor, and then sending that output to the learner to obtain an estimate. Both the preprocessors and learners are from the WEKA [9] data mining toolkit.

Multiple methods are used within the literature to assess the accuracy of estimation across a dataset. As this paper seeks to evaluate these different measures, multiple measures of error will be used and compared. In addition, error measures that are a synthesis of multiple error measures will be assessed to see if collective error measures are more stable.

These different error measures will be compared using paired Mann-Whitney Wilcoxon statistical tests to determine which algorithms perform better and have a significantly different distribution. Algorithms can win, tie (if they have similar distributions), or lose with each possible comparison. Algorithms will be ranked on win and loss measures from the comparison.

The ranking will also be done across multiple datasets, to provide a broader reference as well as potentially gaining information about the datasets themselves. Which algorithms perform well on a given dataset could be indicative of that datasets terrain.

This section will discuss the preprocessors, learners, and error measures used as well as the data sets which they were used upon.

## 3.1 Preprocessors

Before being passed to the learners, the data was run through a preprocessor. Some preprocessors change the values of the data, and others change the shape of the dataset by converting it in to a representative model. The specific preprocessors used will be discussed below.

### *3.1.1 None (none)*

The data is passed to the learner without any preprocessing performed.

### *3.1.2 Logarithmic (log)*

The features in the data are replaced with the natural log of their value. This reduces the distance between features, effecting many of the learners the data is sent to.

### *3.1.3 Equal Frequency Discretization (freq3bin, freq5bin)*

The numeric data is discretized in to a number of bins, with each bin having an equal number of items, or frequency. The data is put in to bins based on numeric value. For example, a 2-bin frequency discretization on the data
$\{1, 4, 2, 8, 3, 9\}$
Would produce a bin $\{1, 2, 3\}$ and a bin $\{4, 8, 9\}$. Equal Frequency Discretization is performed using 3-bins and 5-bins in this experiment

### *3.1.4 Equal Width Discretization (width3bin, width5bin)*

The numeric data is discretized in to a number of bins, with each width having an equal width of starting and ending values contained. The width of each bin is computed as:
$\frac{MaxValue-MinValue}{NumberofBins}$
To provide an example,
$\{1, 2, 3, 4, 8, 9\}$
passed through a 2-bin Equal Width Discretization would produce a bin containing $\{1, 2, 3, 4\}$ and a bin containing $\{8, 9\}$. Equal Width Discretization is done in this experiment with 3 and 5 bins.

### *3.1.5 Normalization (norm)*

Each numeric entry in the data is replaced with a normalized value, computed as:
$\frac{Value-MinValue}{MaxValue-MinValue}$

### *3.1.6 Principal Component Analysis (pca)*

Principal Component Analysis performs a transformation on the features of a dataset. It selects a features with a large enough variance and adds them to the available features. Finally, when no features meet the variance threshold, it creates an additional feature that is orthogonal to the area defined by the previous features included.

### *3.1.7 BAMBOO*

All numerics are logged, and the highest range of variance (standard deviation) for each column of data or attribute is found. Rows are removed, that are part of the highest ranges until the number of rows remaining is: $\sqrt[2]{n}$ Where n is the length of the rows.

## 3.2 Learners

After the data has been preprocessed, it is passed to a learner which makes an estimate on the effort required.

### 3.2.1 Simple Linear Regression (SLReg)

Simple Linear Regression applies n-dimensional linear regression on the data, attempting to determine a correlation of attributes that generate a given effort value. The instance to be tested is then placed along the regression, to estimate for its effort value.

### 3.2.2 Partial Least Squares Regression (plsr)

Partial Least Squares Regression separates the predicting features and the target feature of effort. It treats both as separate matrices, and tries to find a direction in the space of the predicting features that accounts for the target features. This direction serves as a regression model, on which the unknown project can be placed an an estimate obtained.

### 3.2.3 $k$-Nearest Neighbor (1NN, ABE0)

The $k$-Nearest Neighbor finds the $k$ projects in the data that are closest to the unknown projects, and then reports the mean of the projects efforts as an estimate for the new projects effort. In this experiment, $k$ is set to $1$ and $5$ (*ABE0*) for two different runs.

### 3.2.4 Zero R (ZeroR)

ZeroR takes the mean of all effort values in the historical instances and reports it as the estimate.

### 3.2.5 Neural Net (nnet)

A multilayer perceptron is applied on the historical instances, which attempts to create a network of operations on the target feature values that produce an estimate for the target feature. This is done by a stochastic search process in which an initial network is creates, and then a change is made in that network. If error was reduced by the change, the new network is kept and a change is made on it. Otherwise, the previous network is kept. After a number of iterations, a stopping criteria is reached and a final network is produced. The unknown project is put in to the inputs of this network, and the output is reported as the estimate.

### 3.2.6 Decision Stump (DecisionStump)

A decision stump is a decision tree with only a single level. A single attribute is used to predict for the effort.

### 3.2.7 KStar (KStar)

Similar to K-Nearest Neighbors, KStar is another instance based learner. It uses an entropic distance measure rather than euclidean distance.

### 3.2.8 BAMBOO

BAMBOO can be used as a learner as well, hopefully under the condition that it was also used as the preprocessor or some other form of row reduction was used as a preprocessor because it would be very time consuming otherwise. As a learner it takes a test instance and arranges all points in a dataset by their distance to the test data point using euclidian distance. It then takes the k nearest neighbors, skipping over one with a 33

## 3.3 Error Measures

For each dataset, leave one out analysis is performed. In this way, each instance in the dataset is removed to be tested under all preprocessor and learner combinations available, and the estimates stored. Once all estimates have been found, collective error measures are gathered for each combination of preprocessor and learner on each dataset. The error measures used are detailed below.

### 3.3.1 Mean Absolute Residual (MAR)

The absolute residual error of an estimate is computed as:
$|actual - predicted|$
After the absolute residuals have been calculated across a dataset, their mean is taken and reported for the error value.

### 3.3.2 Mean Magnitude of Relative Error (MMRE)

The magnitude of relative error is calculated across a dataset, computed as:
$\frac{|actual-predicted|}{actual}$
After the magnitude of relative error for each instance in the dataset is computed, their mean is reported.

### 3.3.3 Mean Magnitude of Error Relative to the Estimate (MMER)

The magnitude of relative error is computed, but in contrast to MRE it is computed relative to the estimate, as follows:
$\frac{|actual-predicted|}{predicted}$
After the MER is computed for each instance in the dataset, their mean is computed and reported.

### 3.3.4 Median Magnitude of Relative Error (MDMRE)

As MMRE, but the median of the MRE values across the dataset is computed and reported.

### 3.3.5 Pred25

The number of instances in a dataset whose predicted value had an MRE score of less than 25% are divided by the number of total instances, and reported as the Pred25 score.

### 3.3.6 Mean Balanced Relative Error (MBRE)

Balanced relative error is computed as :
$\frac{|actual-predicted|}{dividing}$
Where the dividing term is the smaller of the actual or predicted term. Once the BRE scores have been computed for each instance in the dataset, their mean is computed and reported.

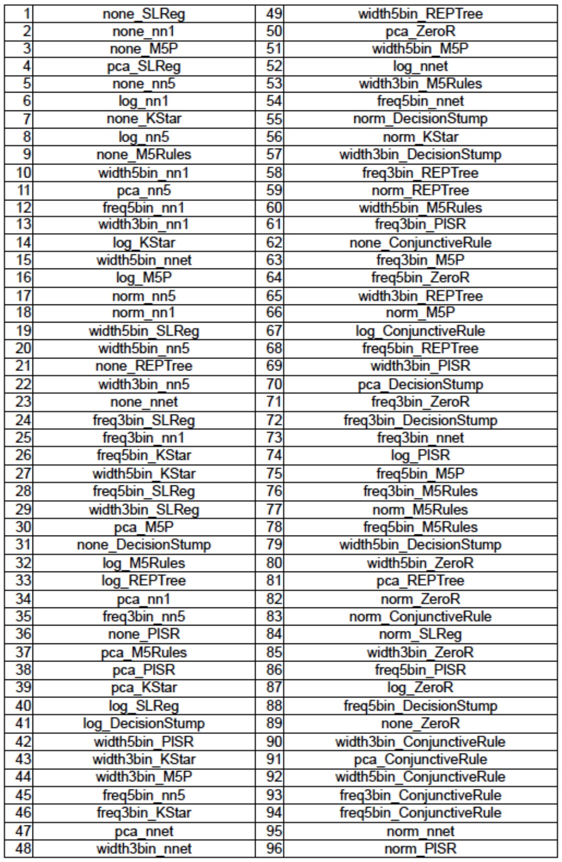
### 3.3.7 Mean Inverted Balanced Relative Error (MIBRE)

Inverted balanced relative error is computed as :
$\frac{|actual-predicted|}{dividing}$
Where the dividing term is the larger of the actual or predicted term. Once the IBRE scores have been computed for each instance in the dataset, their mean is computed and reported.

## 3.4 Data Sets

The datasets used in this experiment were obtained from the PROMISE data repository [1], which provides freely available software engineering data from real world projects. The COMBA software platform used could not handle discrete elements in the data, so discrete elements were removed from the data before being sent to the data preprocessor. The information about each dataset, after removing discrete elements, is provided.

| Dataset | Features | Instances |
|---|---|---|
| Albrecht | 8 | 24 |
| China | 18 | 499 |
| Cocomo81 | 17 | 63 |
| Cocomo81e | 17 | 28 |
| Cocomo81o | 17 | 21 |
| Cocomo81s | 17 | 11 |
| Desharnais | 11 | 81 |
| DesharnaisL1 | 11 | 46 |
| DesharnaisL2 | 11 | 25 |
| DesharnaisL3 | 11 | 10 |
| Finnish | 8 | 38 |
| Kemerer | 7 | 15 |
| Maxwell | 27 | 62 |
| Miyazaki94 | 8 | 48 |
| Nasa93 | 17 | 88 |
| Nasa93_center_1 | 17 | 12 |
| Nasa93_center_2 | 17 | 37 |
| Nasa93_center_5 | 17 | 39 |
| SDR | 24 | 24 |
| Telecom1 | 3 | 18 |



| | | | |
|---|---|---|---|
| 1 | none_SLReg | 49 | width5bin_REPTree |
| 2 | none_nn1 | 50 | pca_ZeroR |
| 3 | none_M5P | 51 | width5bin_M5P |
| 4 | pca_SLReg | 52 | log_nnet |
| 5 | none_nn5 | 53 | width3bin_M5Rules |
| 6 | log_nn1 | 54 | freq5bin_nnet |
| 7 | none_KStar | 55 | norm_DecisionStump |
| 8 | log_nn5 | 56 | norm_KStar |
| 9 | none_M5Rules | 57 | width3bin_DecisionStump |
| 10 | width5bin_nn1 | 58 | freq3bin_REPTree |
| 11 | pca_nn5 | 59 | norm_REPTree |
| 12 | freq5bin_nn1 | 60 | width5bin_M5Rules |
| 13 | width3bin_nn1 | 61 | freq3bin_PISR |
| 14 | log_KStar | 62 | none_ConjunctiveRule |
| 15 | width5bin_nnet | 63 | freq3bin_M5P |
| 16 | log_M5P | 64 | freq5bin_ZeroR |
| 17 | norm_nn5 | 65 | width3bin_REPTree |
| 18 | norm_nn1 | 66 | norm_M5P |
| 19 | width5bin_SLReg | 67 | log_ConjunctiveRule |
| 20 | width5bin_nn5 | 68 | freq5bin_REPTree |
| 21 | none_REPTree | 69 | width3bin_PISR |
| 22 | width3bin_nn5 | 70 | pca_DecisionStump |
| 23 | none_nnet | 71 | freq3bin_ZeroR |
| 24 | freq3bin_SLReg | 72 | freq3bin_DecisionStump |
| 25 | freq3bin_nn1 | 73 | freq3bin_nnet |
| 26 | freq5bin_KStar | 74 | log_PISR |
| 27 | width5bin_KStar | 75 | freq5bin_M5P |
| 28 | freq5bin_SLReg | 76 | freq3bin_M5Rules |
| 29 | width3bin_SLReg | 77 | norm_M5Rules |
| 30 | pca_M5P | 78 | freq5bin_M5Rules |
| 31 | none_DecisionStump | 79 | width5bin_DecisionStump |
| 32 | log_M5Rules | 80 | width5bin_ZeroR |
| 33 | log_REPTree | 81 | pca_REPTree |
| 34 | pca_nn1 | 82 | norm_ZeroR |
| 35 | freq3bin_nn5 | 83 | norm_ConjunctiveRule |
| 36 | none_PISR | 84 | norm_SLReg |
| 37 | pca_M5Rules | 85 | width3bin_ZeroR |
| 38 | pca_PISR | 86 | freq5bin_PISR |
| 39 | pca_KStar | 87 | log_ZeroR |
| 40 | log_SLReg | 88 | freq5bin_DecisionStump |
| 41 | log_DecisionStump | 89 | none_ZeroR |
| 42 | width5bin_PISR | 90 | width3bin_ConjunctiveRule |
| 43 | width3bin_KStar | 91 | pca_ConjunctiveRule |
| 44 | width3bin_M5P | 92 | width5bin_ConjunctiveRule |
| 45 | freq5bin_nn5 | 93 | freq3bin_ConjunctiveRule |
| 46 | freq3bin_KStar | 94 | freq5bin_ConjunctiveRule |
| 47 | pca_nnet | 95 | norm_nnet |
| 48 | width3bin_nnet | 96 | norm_PISR |

**Figure 1: All permutations of data miners and learner pairs, ordered from least amount of loss measures to the most.**

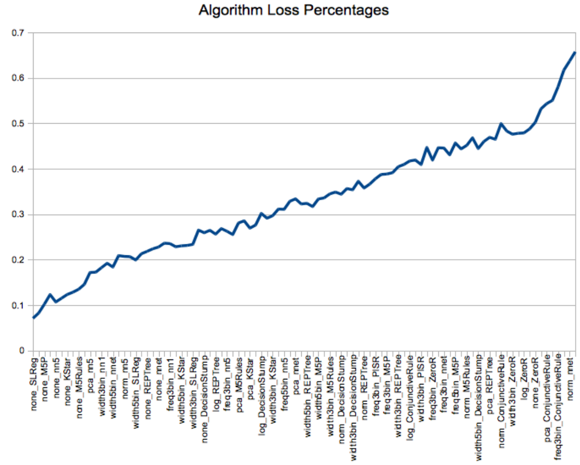


**Figure 2: The 96 algorithms sorted by percentage of losses.**

# 4. RESULTS

The results here follow the procedure outlined in the experimental design, reporting the rankings of the different learner combinations. The rankings are generated by first comparing a learner with each other learner across the 7 different error measures using a paired Wilcoxon ranked test for significance. If the test shows significance, the descriptive statistic such as mean or median is compared and the algorithm with closer predictions is marked a winner while the other is marked a loser. If both methods are equal or the test does not show significantly different distributions, a tie is reported.

The percentage of possible losses is computed for each dataset by comparing how many times an algorithm lost to how many times it could have lost. With 96 algorithm combinations and 7 error measures, an algorithm can lose up to 665 times on a given dataset. The sum of the number of losses on each dataset is added up and divided by the total number of possible losses across all datasets. This measure is referred to as percentage of losses or loss percentage. Figure **??**

Algorithms with a lower percentage of losses are ranked higher for the sort order.Figure 1 In the ranks, none_SLReg is the best performing algorithm while norm_PlSR is the worst performing. These rankings are consistent with an earlier experiment performed in a different simulation environment.

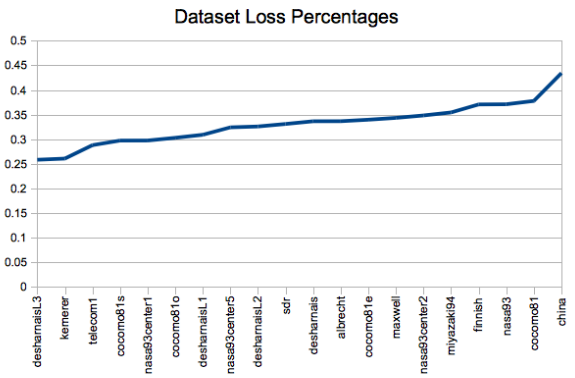


**Figure 3: The 20 datasets sorted by percentage of losses.**

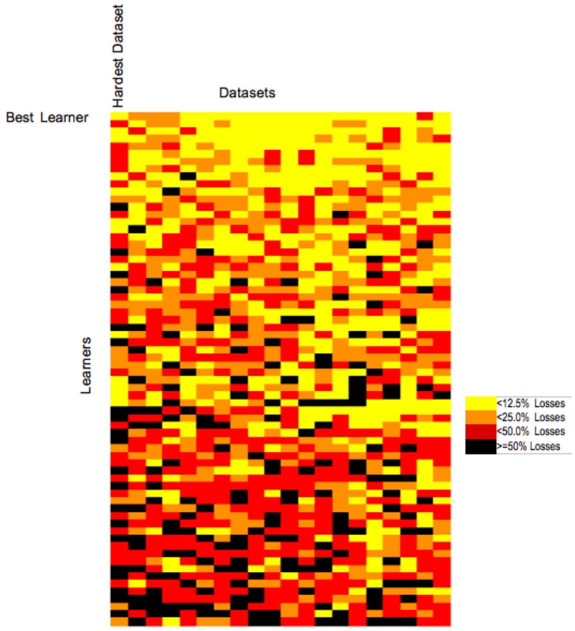


**Figure 4: The 20 datasets and 96 algorithms, with order determined by rankings in previous figures.**

# 5. CONCLUSION

The results in the experiment are different to the previous study using COMBA. They suggest more homogeneity across the datasets, and do not show data that is superior for learning.

This paper is part of an ongoing research, so future works involving COMBA will be provided. Not all of them are in the scope of the final project implementation, so they will be listed in order of priority to the current research project.

## 5.1 Future Work

There are still many open questions about evaluation methods for ranking search based software engineering algorithms. A few ways in which the experiment could be expanded are provided below.

### *5.1.1 More Algorithms, More Datasets*

The 52 algorithms covered in this paper are only a small subset of the available algorithms for search based software engineering. While data mining and artificial intelligence are relatively young fields of computer science, they have already generated hundreds of algorithms which can be combined in thousands of ways. The combination of algorithms approach could be extended to include a more diverse range of algorithms, and scalability of the results could be assessed. Additionally, there are more datasets available for effort estimation in the PROMISE data repository. These additional datasets could be included and tested, to see if initial results hold across a larger range of datasets.

The current COMBA coding system has an open-source implementation in which algorithms are called from shell scripts, allowing algorithms to be collected from multiple languages across multiple environments. This allows for the rapid expansion of the system by adding already implemented and available methods.

This task is suggested primarily because of the difference in results between versions of COMBA. The MWW tests performed for algorithm comparison are biased based on the performance of the algorithms included in the COMBA system. It is possible that there were not a large enough number of poor or high performance algorithms, both of which could account for the difference in results.

### *5.1.2 Other Fields of Search-Based Software Engineering*

This paper only examined the method of effort estimation, while many algorithms exist for other aspects of project planning. The combination of algorithms approach could be applied to tasks such as defect prediction and cost optimization. On a broader scale, the combination of algorithms method could be used to evaluate algorithms from domains other than search based software engineering.

### *5.1.3 Algorithm Optimization*

Many algorithms used within the COMBA system, such as Neural Nets and Principle Component Analysis can be run using a variety of different settings. In addition to more algorithms, algorithms can be run with different specifications. Many times, specifications for algorithms are decided using ëngineering judgment ̧, rather than empirical methods. Sometimes this involves citing another paper in which a particular setting was used, even if that paper did not test other settings. The diverse range of data sets tested for might show that certain settings tend to perform better, and provide a basis for further research using those methods.

Additionally, the different specifications for algorithms can be tested and compared to one another. In this paper, several algorithms (CART, kNN, discretization) were tested with different values and compared. The program can control setting optional values, and use a search process to find optimal values for these settings through repeated runs and comparisons.

This does not only need to be used on existing methods. COMBA also serves as an environment for creating new algorithms. Supposing an algorithm is created as a baseline and run on the available datasets, future runs of the algorithm with changes made can be compared to the previous states, allowing tests to see if changes have improved performance or not.

### *5.1.4 Rank On Other Dimensions*

It should be noted that only the error measures provided were used to assess the given algorithms. There are many other factors on which an algorithm can be evaluated, such as Big-O notation or empirical runtime results. Future tests could collect the time taken to evaluate each dataset, and provide this along with error measures. This could also be used as a complexity measure, to assess if complex and time-consuming algorithms increase performance or not.

### *5.1.5 Real Time Readjustment*

A combination learning approach could be applied in an incremental fashion on data whose results depend on predictions made by a given algorithm. For example, suppose a real-time simulation of a company is performed using estimates produced by algorithms which performed well on previous data. Changes in performance can be taken in to account, so that multiple time steps are available and can be treated as different datasets. This allows back propagation for algorithm recommendation and evaluation, and for certain applications can uncover more details about the underlying assumptions of data generation.

### *5.1.6 Domain Specific Knowledge Acquisition*

One aspect of the datasets has a real world value corresponding to it. Other papers have proposed feature weighting, in which certain aspects of a project are more important than others for predicting effort. Domain inspecific ways, such as genetic algorithm optimization, have been proposed to find the best set of feature weights. A broad approach could look at the application of certain features across different datasets. For example, lines of code is generally regarded to be a good indicator of project effort, though it itself often requires estimation to obtain. This null hypothesis could be tested, by stripping lines of code from the dataset and viewing the results. This approach could be made more general by removing features to test their effects on estimation accuracy using different algorithms.

### *5.1.7 Experiment Verification and Validation*

COMBA provides an environment to reproduce results published in other papers. Experiments which relied on a limited number of datasets or algorithms to compare to could be reproduced. A possibility would be reproducing multiple experiments in one paper, and noting if the assumptions in the initially published result held upon more intensive testing.

More ambitiously, the COMBA system is freely available and could be recommended for use. In this way, researchers in other areas could download COMBA and quickly validate their results. There are more ways to expand this project then are feasible given the time-scale, so this approach provides a system which will build itself. The more people using the system, the stronger the results produced from it become as new algorithms and datasets become contributed to the COMBA system.

### *5.1.8 Performance Analysis*

No reason was given as to why the algorithms which performed well did so. A further study could be done in which it was observed for which type of datasets algorithms performed well, and the properties of the data that effect their performance. This has the benefit of not requiring domain specific software engineering data. Any n-dimensional space could be analyzed, as many of the algorithms used in search-based software engineering do not make domain specific assumptions about the data. If COMBA is run on a large number of algorithms and datasets, a large database of results becomes available. This makes possible rapid testing and verification by comparing to existing results stored from previous runs.

These results could be used for suggesting algorithms given a dataset. Similarity measures between datasets could be evaluated, and high-performing algorithms from similar datasets could be suggested without having to test them on the dataset. The results of the recommendation system could then be compared to the actual results of running the possible algorithms. While neural nets and other stochastic optimization processes are typically associated with poor runtimes, a large database provides a terrain for optimization in which no complex algorithms need to be run. The results would already be precomputed, so verification could be done within reasonable time bounds. A study could also be done where results are incrementally added to the database, and the optimization effectiveness is based on number of datasets. It would be interesting to see if over fitting occurs with a few datasets, or if that too many datasets could flood the optimizer so that results are too general to be applicable.

## 6. REFERENCES

[1] Promise repository of empirical software engineering data. `http://promisedata.org/repository`, 2010.

[2] M. Harman. The current state and future of search based software engineering. *Future of Software Engineering*, 0:342–357, 2007.

[3] M. Harman and B. F. Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833 – 839, 2001.

[4] M. Jorgensen and S. Grimstad. Over-optimism in software development projects: The winner#146;s curse. pages 280–285, 2005.

[5] J. Keung, E. Kocagunelli, and T. Menzies. Where is the best effort estimator?

[6] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28:721–734, 2002.

[7] K. Molokken and M. Jorgensen. A review of software surveys on software effort estimation. In *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*, pages 223 – 230, sept.-1 oct. 2003.

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

[9] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2nd edition edition, 2005.

## APPENDIX

The authors of this paper would like to note that it is a continuation of an existing study by Jacky Keung, Ekrem Kocagunelli, and Tim Menzies. The paper describing that study is not yet published or publicly available, so no link or reference can be provided at this time.