Evaluation bias in effort estimation

Omid Jalali
West Virginia University

Evaluation Bias in Effort Estimation

Omid Jalali

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2008

Keywords: COCOMO, effort estimation, row pruning, evaluation, Mann-Whitney U test

Abstract

Evaluation Bias in Effort Estimation

Omid Jalali

There exists a large number of software effort estimation methods in the literature and the space of possibilities [54] is yet to be fully explored. There is little conclusive evidence about the relative performance of such methods and many studies suffer from instability in their conclusions. As a result, the effort estimation literature lacks a stable ranking of such methods.

This research aims at providing a stable ranking of a large number of methods using data sets based on COCOMO features. For this task, the COSEEKMO tool [46] was further developed into a benchmarking tool and several well-known effort estimation methods, including model trees, linear regression methods, local calibration, and several newly developed methods were used in COSEEKMO for a thorough comparison. The problem of instability was further explored and the evaluation method used was identified as the cause of instability. Therefore, the existing evaluation bias was corrected through a new evaluation approach, which was non-parametric. The Mann-Whitney U test [42] is the non-parametric test used in this study, which introduced a great amount of stability in the results. Several evaluation criteria were tested in order to analyze their possible effects on the observed stability.

The conclusions made in this study were stable across different evaluation criteria, different data sets, and different random runs. As a result, a group of four methods were selected as the best effort estimation methods among the explored 312 combinations of methods. These four methods were all based on the local calibration procedure proposed by Boehm [4]. Furthermore, these methods were simpler and more effective than many other complex methods including the Wrapper [37] and model trees [60], which are well-known methods in the literature.

Therefore, while there exists no single universal best method for effort estimation, this study suggests applying the four methods reported here to the historical data and using the best performing method among these four to estimate the effort for future projects. In addition, this study provides a path for comparing other existing or new effort estimation methods with the currently explored methods. This path involves a systematic comparison of the performance of each method against all other methods, including the methods studied in this work, through a benchmarking tool such as COSEEKMO, and using the non-parametric Mann-Whitney U test.

Dedication

To My Family

Acknowledgments

I would like to thank my advisor, Dr. Tim Menzies, for introducing me to software engineering research, teaching me a great deal about data mining and effort estimation, and introducing me to outstanding researchers at the ICSE conference, allowing me to learn more than possible. He gave me the motivation to start this research, and despite his extremely busy schedule, has kept me motivated all along. I am and will always be grateful for everything he has done for me.

I am thankful to the Lane Department and West Virginia University. I have had the unique opportunity of learning about computer science in this department and about many other branches of science at West Virginia University. I would like to thank Dr. John Atkins for his thorough courses in databases and his help as my academic advisor, Dr. Elaine Eschen for the great theory courses that made me more interested every day and for her patience in answering my numerous questions, Dr. Tim Menzies for his data mining course that taught me invaluable things about this subject, and Dr. Jim Mooney for his great operating system courses.

I would like to thank Dr. Jairus Hihn from NASA's JPL, Dr. Afzel Noore, and Dr. Tim Menzies for serving on my committee. I would like to thank Dr. Hihn for his support and statistical advices. I would like to thank Karen Lum from JPL and Dr. Hihn for collaborating on our papers.

I want to thank Dan Baker, a great friend and colleague, who wrote Cocomin and provided a unique opportunity to test this work at JPL. I also thank my friends Zach Milton, Brian Sowers, Nathan Moore, Phillip Green, Ous Elrawas, and DJ Boland for their help, support, and humor.

Finally, I thank my parents and my family for their constant support throughout my education.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software effort estimation is, as its name suggests, the task of estimating the amount of effort required to develop new software. It is essential for software developers to estimate a new project's effort in order to make a decision about developing or not developing a software project. In large projects, it is critical that all resources are planned ahead of time and money is allocated to the project. In many cases, the success or failure of a company or organization may depend on such a software project.

According to Boehm, software effort estimates are often wrong by a factor of four [4]. Other studies have reported larger errors (of 85% to 772% with many in the range of 500% to 600%) when comparing different software models [33]. A recent example is the NASA's CLCS system, which was cancelled in 2003. The initial estimate for the project was 206 million dollars, which was increased to between 488 million and 533 million dollars, more than twice the initial estimate. On cancellation, approximately 400 developers lots their jobs [70].

There are other studies exploring the problem of overrunning projects. Molokken and Jorgensen report that 60% to 80% of projects "encounter effort and/or schedule overruns [51]." According to Standish Group's Chaos Report, "31.1% of projects will be cancelled before they ever get completed. Further results indicate 52.7% of projects will cost 189% of their original estimates [72]," meaning that there is a 89% overrun in these projects.

This could mean that the remaining projects may have to be cancelled or postponed in order to cover the costs of these projects. Although Jorgensen and Molokken briefly question such numbers [28], they report an average of 30% to 40% overrun [28, 51]. There are other studies reporting several cancelled, delayed, and overrun projects. The lesson from these studies is to search for methods that can reduce the error in effort estimation and produce reliable and accurate estimates.

Researchers have developed several software models and numerous effort estimation methods, some of which can be found in [54]. This has caused two problems. One is that, each organization capable of collecting historical data has used a different model to collect data and most likely kept it private. Therefore, there is usually no access to data in order to research certain models' performance. This has led this study to use the COCOMO model developed and later improved by Boehm [4, 7] with two publicly available data sets.

Another problem is that, although there are numerous studies on different effort estimation methods (for example, Jorgensen and Shepperd's study [30] identified 304 software cost estimation papers), there are few empirical studies comparing these methods, and more importantly, reporting stable conclusions. There are, however, few studies that compare different software models, such as COCOMO II, SEER-SEM, PRICE-S comparison in [40], REVIC, SASET, PRICE-S, SEER-SEM, SLIM, SOFTCOST, CHECKPOINT, COCOMO II, and SAGE comparison in [19], and SLIM, COCOMO, Function Points, and ESTIMACS comparison in [33]. Although these studies are helpful in the sense that they provide a unique opportunity to compare the performance of these models (mostly without any notable publicly available data), they also add to the confusion of what model or method to use since they either report no clear winner [19, 40] or evaluate the results differently and offer different conclusions based on the method of evaluation [33].

It is clear that unless the numerous effort estimation methods are ranked systematically, the confusion will remain, and will possibly get worse as new methods are added. This study aims at such a task. Different combinations of the representatives of several well-known groups of methods are ranked in this research in an effort to introduce a framework for such rankings in the future, as well as reporting the best performing methods among the 312 combinations of methods studied here.

The importance of the results of this research is not only in the systematic method of comparison offered, but is also in the stability of the conclusions made. The evaluation method used in this research plays a key role in illustrating this stability, which is the main feature missing from many studies [19, 33, 40, 46, 67]. As an example, Shepperd and Kadoda's study attempted at "comparing four contrasting techniques of stepwise regression, rule induction, a form of case-based reasoning, and neural nets [67]." However, they were unable to show conclusive results about these methods and their possible rankings relative to each other. It should be noted that their experiments used simulated data (first proposed by Pickard et al. [56] where they explored simulated non-normal data) and were repeated across different data sets and used different sampled training sets in different runs. Although no conclusive results were offered, their study contained some elements of a systematic comparison, such as different data sets and runs.

The experiments in this work have the same goal of ranking methods and use a similar approach. The COSEEKMO tool [46] is further developed to incorporate several different methods, including model trees [60], linear regression methods, case-based reasoning methods [54, 69] (using estimation by analogy), thorough column pruning methods such as Wrapper [37], local calibration [4] methods, as well as newly developed column pruners [2] and row pruners (Section 3.2.3).

There is an evaluation bias in every study of effort estimation methods, due to the selection of specific tests to compare these methods. This study will introduce a non-parametric evaluation method called the Mann-Whitney U test (Section 3.4.2), which does not assume an underlying distribution and hence, is less affected by the nature of the experiments. In order to further reduce the bias in evaluation, three different evaluation criteria will be used to measure the performance of each method. Also, different data sets (and their subset) and different random runs of the experiments will be used to compare all methods involved.

The results of this research show a great amount of stability in the conclusions (unlike other prior studies) and report four effort estimation methods as the best performing methods among the 312 combinations of methods explored. These methods are all based on the local calibration method developed by Boehm [4]. A row pruning method developed in this work is used in two of these four methods. These four methods were recently implemented in a tool called 2CEE for NASA's Jet Propulsion Laboratory and have shown outstanding results [2].

1.1 Contributions of This Thesis

This thesis has made the following contributions to both the literature and practice:

- A new framework for comparing and ranking effort estimation methods.
- One of the few studies that compared the representatives of many effort estimation methods in a large scale, through 312 combinations of such methods, and reported stable results.
- Reducing the search space for effort estimation methods to a small group of four top performing methods.
- Exploring the evaluation bias in the comparison of effort estimation methods and aiming at reducing its effects in such comparisons.
- Suggesting the Mann-Whitney U test as a non-parametric evaluation method in effort estimation, which reduces the instability in the results. This suggestion was recently taken into consideration by Baker [2] and showed similar stability in the results.
- A simple row pruning method, called Locomo and based on the local calibration procedure and the nearest neighbor algorithm, which performs well and is reported in the top four

performing methods in this study.

- Industrial implementation of Locomo in the 2CEE tool developed for NASA's Jet Propulsion Laboratory.

1.2 Structure of This Document

The outline of the remaining chapters is as follows:

- Chapter 2 explores related studies and provides a background of the current research.
- Chapter 3 gives the details of the experiments performed in this study and introduces tools and methods used or developed for this research. It also offers the results and the discussion of these experiments.
- Chapter 4 summarizes this study and offers conclusions. It also describes possible future works.
- In addition, there is an appendix containing the instructions for the experimental tool used in this study.

Chapter 2

Background and Related Work

This chapter provides background information about this research. It is important to understand the previous work and existing problems in this area before presenting the experiments and results since this study attempted to solve such problems. The effort estimation concept and different approaches to effort estimation are discussed in Section 2.1. Section 2.2 focuses on the COCOMO model used in this study and briefly discusses other models. The concepts of pruning columns and row are provided in Sections 2.3 and 2.4 respectively. In this study, these pruners are considered pre-processors to learners, discussed in Section 2.5. Different evaluation methods are discussed in Section 2.6 and the instability of such evaluations in effort estimation is discussed in Section 2.7. Finally, a summary of the current common problems in the effort estimation literature is presented in Section 2.8.

2.1 Effort Estimation

Effort estimation for software projects serves a major role in the success or failure of such projects, especially in the case of large projects. As mentioned in Section 1, inaccurate estimates are reported widely in the literature and have caused the cancellation of overrunning projects. Boehm

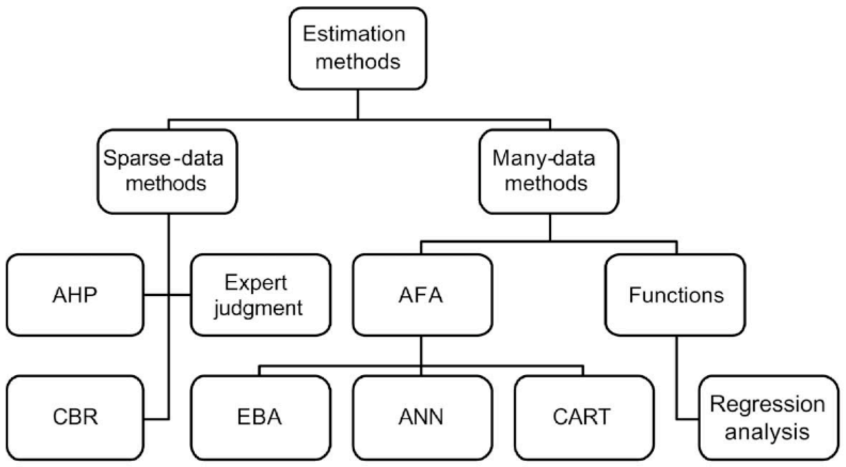

Figure 2.1: Classification of Effort Estimation Methods [54].

and Papaccio's study about software costs concludes that "understanding and controlling software costs is extremely important [6]." This section provides an overview of models and methods that could help in estimating such costs as accurately as possible, which could simplify the process of controlling software costs as [6] mentions.

Effort estimation methods can be classified in different ways. Section 2.1.1 takes a look at one such classification, which focuses on the type of effort estimation methods, broadly categorized based on the amount of historical data they need. The remaining sections discuss this classification in detail.

2.1.1 Classification of Software Effort Estimation Methods

Myrtveit et al. published a study where they classified the current effort estimation methods in two groups of *sparse-data* and *many-data* methods [54]. This is a classification based on the availability of historical data. According to [54], "sparse-data methods are estimation methods requiring few or no historical data" while many-data methods need more historical data for effort estimation. Each group contains different types of methods.

Sparse-data methods include:

- Analytic Hierarchy Process (AHP),
- Expert Judgment, and
- Automated Case-Based Reasoning (CBR).

Many-data methods are subdivided into functions and arbitrary function approximators (AFA). According to [54], "arbitrary function approximators do not make any assumptions regarding the relationship between the predictor and response variables" while functions assume otherwise. AFA's include:

- Estimation by Analogy (EBA),
- Artificial Neural Networks (ANN), and
- Classification and Regression Trees (CART).

Myrtveit et al. explore some of these methods. Overall, there are 7 groups of effort estimation methods based on this classification. However, EBA may be considered a kind of CBR. As they mention, "there is a spectrum between sparse and many-data methods" and CBR may belong to both depending on how it is used. "If CBR is used to identify the closest case, it is a many-data method. EBA is an example of this use of CBR. [54]." Therefore, they view EBA as a type of CBR and the same approach is used in Section 2.4 to study both methods for row pruning.

2.1.2 Analytic Hierarchy Process

Analytic Hierarchy Process (AHP) [63], in general, is a technique for dealing with complex decisions by determining the right decision suited for each case, or criteria, rather than using a generic decision. Each problem in analyzed through the hierarchy of smaller problems that are easier to solve. In essence, this is a structured problem-solving technique that deals with sub-problems concurrently and makes a decision based on the relative ranking of the solutions for each sub-problem.

AHP can be applied to effort estimation in various ways. As Shepperd and Cartwright suggest, "an example would be to utilize the hierarchical structure of AHP to predict effort by decomposing the problem into a number of criteria which all contribute to actual effort [65]." They use AHP (as well as a pairwise comparison technique) in the study of their sparse-data method and report promising results. AHP, as a multicriteria decision-making technique, may be applied to effort estimation, which uses a single criterion, namely effort. Moreover, other contributing factors to effort such as function points and other information about the project can extend this hierarchy [65].

2.1.3 Expert Judgment

According to Jorgensen, an estimation strategy is categorized as expert estimation, or expert judgment, when "the estimation work is conducted by a person recognized as an expert on the task, and that a significant part of the estimation process is based on a non-explicit and non-recoverable reasoning process [27]." Clearly, the extent to which the expert follows guidelines in this process can affect the successful reproduction of the results of this process. Nonetheless, the expert may or may not follow any instructions or use the historical data.

There are twelve "best practice" expert estimation principles suggested by Jorgensen [27] based on empirical evidence:

1. Evaluate estimation accuracy, but avoid high evaluation pressure.
2. Avoid conflicting estimation goals.
3. Ask estimators to justify and criticize their estimates.
4. Avoid irrelevant and unreliable estimation information.
5. Use documented data from previous development tasks.
6. Find estimation experts with relevant domain background and good estimation records.

7. Estimate top-down and bottom-up, independently of each other.

8. Use estimation checklists.

9. Combine estimates from different experts and estimation strategies.

10. Assess the uncertainty of the estimate.

11. Provide feedback on estimation accuracy and task relations.

12. Provide estimation training opportunities.

According to Jorgensen [27], principles 1 through 5 “reduce situational and human biases,” principles 7 through 10 “support the estimation process,” and principles 11 and 12 “provide feedback and training opportunities.” Jorgensen also cautions that there is “still a need for more knowledge about how to apply them in various software estimation situations. [27].” Baker [2] studied these principles and implemented seven of the twelve best practices in the 2CEE tool designed for NASA’s Jet Propulsion Laboratory.

Expert judgment approaches are widely used for effort estimation in the industry [27, 30]. Jorgensen [27] reports five studies were 62%, 72%, 83%, 84%, and 86% of the effort estimation in a wide range prominent organizations and companies were performed using expert judgment.

In their review of effort estimation studies, Jorgensen and Shepperd claim that “more formal estimation techniques” or model-based methods “has been somewhat erratic to date, has not documented higher accuracy than expert judgment, and that expert-based estimation approaches are, by far, the most common used approaches by the software industry [30].” Although, according to [30], model-based methods have not documented higher accuracies than expert-based methods, Jorgensen mentions that “it is not possible to conclude that expert estimation or estimation model, are more accurate” and offers an observation where expert-based methods seem to be more accurate “when there are important domain knowledge *not* included in the estimation models, when the estimation uncertainty is high as a result of environmental changes not included in the model, or when simple estimation strategies lead to relatively accurate estimates [27].” Therefore, there is no clear winner in the comparison of the two approaches in effort estimation. Jorgensen offers empirical results where he compared 15 studies that compare expert-based estimation with model-based estimation. Among these studies, five were in favor of expert-based estimation, five were in favor of model-based estimation, and five did not find any difference [27].

Therefore, although the expert-based estimation is widely used in the industry, the lack of a clear winner between the two approaches has led researchers to study the model-based effort estimation extensively . Another reason for the tendency of researchers toward model-based methods may be the lack of expert judgments in the research community and ease of reproducing model-based methods' results comparing to that of the expert-based methods.

In general, model-based methods, such as the ones in Figure 2.1 (functions, different arbitrary function approximators (AFA) including EBA, ANN, and CART, and even CBR in certain cases) use some algorithm to summarize the historical data and offer estimates for new projects. These methods are discussed in their corresponding sections. Menzies et al. [46] extended the twelve best practices for expert-based estimation to include eight best practices for model-based estimation:

> According to Boehm [4, 7]; Chulani [13, 62]; Kemerer [33]; Stutzke [73]; Shepperd [68]; our own work [11, 12, 45]; and a recent tutorial at the 2006 International Conference of the International Society of Parametric Analysts [24], best practices for model-based estimation include at least the following:
>
> 13. *Reuse regression parameters* learned from prior projects on new projects;
> 14. *Log-transforms* on costing data before performing *linear regression* to learn log-linear effort models;
> 15. *Model-tree* learning to generate models for non-linear relationships;
> 16. *Stratification*, i.e. given a database of past projects, and a current project to be estimated, just learn models from those records from similar projects;

17. *Local calibration*, i.e. tune a general model to the local data via a small number of special tuning parameters;

18. *Hold-out* experiments for testing the learned effort model [45];

19. Assessing effort model uncertainty via the *performance deviations* seen during the hold-out experiments of item #17;

20. *Variable subset selection* methods for minimizing the size of the learned effort model [11,12,34,49].

(These practices were applied or further explored in this study with the exception of performance deviations being rejected and replaced with a non-parametric test. The results are explained in Section 3.6.)

They also included best practices for the combination of expert-based and model-based estimation [46]:

> This separation of model-based and expert-based methods is not a strict division since some practices fall into both categories: e.g. #4 & #20 are similar as are #10 & #19. Also, one way to view model-based methods is that they seek algorithms to make maximal use of #5. Further, some research actively tries to combine the two approaches:
>
> 21. Shepperd's *case-based reasoning* tools [68] explore algorithmic methods for emulating expert analogical reasoning;
>
> 22. Chulani & Boehm's *Bayesian tuning method* [13] for regression models allows an algorithm to carefully combine expert judgment with the available data;
>
> 23. This paper will argue for the use of heuristic *rejection rules* to represent expert intuitions on how to rank different effort models.

(Case-based reasoning was applied and further explored in this study and rejection rules were replaced with a non-parametric test. The Bayesian tuning method can be explored as a future work. The results of the case-based reasoning method are explained in Section 3.6.)

Baker's attempt [2] to integrate as many as seven of Jorgensen's twelve best practices [27] into a tool, which also involves model-based methods emphasizes the suggestions of other researchers [64] to combine expert-based and model-based effort estimation methods and create a unified approach for effort estimation, which could potentially increase the accuracy of the estimates. The Bayesian estimation approach suggested by Chulani et al. [13] is another method to merge "expert prior information with software engineering data," which is especially useful when the data is scarce and incomplete.

2.1.4 Automated Case-Based Reasoning

Case-based reasoning (CBR) is an approach for finding similarities between past projects and a future project and using them in predicting certain features, such as effort, for future projects. In this method, when seeking a solution to a current problem, the problem is matched against other solved problems and their solutions (altogether called cases) in the case base. Solutions to similar problems are then used to find a solution to the current problem, which is tested for success. This new solution may be revised if necessary. The new problem and its solution will be stored as a new case in the case base. In short,

> case-based reasoning has four distinct aspects:
>
> - characterization of cases,
> - storage of past cases,
> - retrieval of similar cases to use as analogies, and
> - utilizing the retrieved case to solve the target case problem, sometimes known as case adaptation [67].

In a general sense, CBR and the methods based on it may seem similar to rule induction algorithms such as C4.5 [61] in that both approaches look for similarities in the data. However, there is a difference in how they use their training data. CBR uses the test instance (current problem) to generalize as it is processing its train instances while a rule induction algorithm finishes training a model before it can use it on a test instance. In other words, according to Shepperd and Kadoda,

> CBR is considered to be fundamentally different from rule induction and regression approaches in that it utilizes specific knowledge of previously solved cases to solve future ones, while the former use generalized knowledge or relationships, respectively. In other words, CBR is model free [67].

2.1.5 Functions

Functions are the first group of many-data methods analyzed by Myrtveit et al. [54] and are considered of the general form $y = Ax^B$. In general, functions relate the features (also called the variables) of a model mathematically. The linear regression method (discussed in Section 3.5.5) is a function, and the only one discussed in [54]. Boehm, who developed the COCOMO model [4] (discussed in Section 2.2.1), also developed the local calibration method, which is a specialized form of linear regression [4, p526-529]. Local calibration is widely used as an effort estimation method, at least in the COCOMO community. Local calibration is further discussed in Section 3.5.5.

There are also other forms of regression, such as the multiple regression approach.

> A multiple regression model can be written as
>
> $$y_t = \beta_0 + \beta_1 x_{t1} + \cdots + \beta_k x_{tk} + \varepsilon_t$$
>
> where $x_{t1} \cdots x_{tk}$ are the values of the predictor (or regressor) variables for the x_{th} observation, $\beta_0 \cdots \beta_k$ are the coefficient to be estimated, ε_t is the usual error term, and y_t

is the response variable for the y_{th} observation [13].

Chulani et al. report the popular classical multiple regression approach as the most commonly used technique for empirical calibration [13]. Their study explores the problems faced by the multiple regression approach during the calibration of COCOMO II [7], an updated version of COCOMO. Their study suggests the Bayesian approach over the multiple regression approach. Nonetheless, regression is widely used in many studies in the effort estimation literature. According to Jorgensen and Shepperd [30], "regression-based estimation approaches dominate" the space of estimation approaches with a total of 49% of the studies published between 1989 and 2004. It should be noted that they also included the methods based on COCOMO in this list.

2.1.6 Estimation by Analogy

Estimation by analogy can be considered a type of case-based reasoning [54] and has become the focus of more studies recently [30]. Among other methods, Boehm suggested estimation by analogy as a method for effort estimation [4]. According to Boehm, "the main strength of estimation by analogy is that the estimate is based on actual experience on a project [4]." Since it is not clear to what degree the previous project represents the current project, and Boehm considered this its main weakness, estimation by analogy should be used with caution. However, Boehm did not explore the possibility of automating this process at that time.

Later, Cowderoy and Jenkins [15] mentioned estimation by analogy as a "widely exercised practice within the software industry" and as a partly automated process. Nonetheless, this process was not fully automated until later. Among other tools, ANGEL software tool described by Shepperd et al. is an automated tool that uses estimation by analogy [69]. In their effort estimation study, Shepperd and Schofield suggested using estimation by analogy as a viable technique in addition to other effort estimation methods [68]. Effort estimation using estimation by analogy and case-based reasoning in general are extensively studied by other researchers as well [1,29,31,32,38,41,52,53,57,58]. Some, however, are skeptical about the application of case-based reasoning in effort estimation in every domain. Delany et al. studied the problems associated with extending the use of CBR for cost estimation to operate across broader domains [16].

2.1.7 Artificial Neural Networks

As their name suggests, artificial neural networks (ANN) are mathematical methods designed based on the neural activities of neurons of living organisms in the nervous system, and more specifically the brain. Similar to the brain, where an input (stimuli) can be related to an output (action), an ANN can model the relations between inputs and outputs.

An application of ANN is in data mining and finding patterns. Effort estimation is also possible using ANN models, although Myrtveit et al. mentioned contradicting studies about the relative performance of ANN models comparing to regression models discussed before [54]. Another study (also discussed in Section 2.7) is the Shepperd and Kadoda's study where they found ANN usually performing worse than stepwise regression (a variant of regression), rule induction, and case-based reasoning [67]. Nonetheless, researchers have used and recently focused more on ANN as an effort estimation method [30].

2.1.8 Classification and Regression Trees

Classification trees were studied both in statistics and in artificial intelligence. The statistics studies by Breiman et al. led to the classification and regression trees (CART) [9] and among other studies in artificial intelligence, classification trees led Quinlan to ID3, which was later extended into C4.5 [61]. In essence, CART creates a classification tree of a group of variables and their split points that most reduce the impurity of the root node, where impurity is measured using a specific function [77].

The application of CART in effort estimation may seem strange due to the presence of a classification tree. However, as Briand et al. explain, this is possible:

> CART examines the data and determines data splitting criteria that will most successfully partition the dependent variable. To build a regression tree the data set is recursively split until a stopping criterion is satisfied. All but the terminal nodes in a tree specify a condition based on one of the variables that have an influence on project effort or any other dependent variable.
>
> After a tree is generated it may be used for effort prediction of a project. To predict the effort for a project, a path is followed through the tree according to the project's specific variable values until a terminal node is reached. The mean or median value of the terminal node may then be used as the predicted value [10].

According to Jorgensen and Shepperd [30], CART is used in effort estimation studies, which is 5% of studies over a span of 15 years. Myrtveit et al. [54] report contradicting studies about the merits of CART comparing to regression models. One of such studies is by Briand et al. [10], where they report contradicting results comparing to their previous studies on the benefits of CART when used alone or in combination with other methods. They conclude that "depending on the structures present in the data, we have to expect variations in performance across data sets [10]." Kitchenham applied CART to the COCOMO data set and reported other simpler methods outperforming it [35]. Kitchenham suggest using an algorithm like CART when there is a large number of data points in the data set.

2.2 Software Effort Estimation Models

There are many software effort estimation models available. This study focuses on COCOMO, discussed in Section 2.2.1. Other effort estimation models are also discussed briefly in Section 2.2.2.

2.2.1 COCOMO

The COnstructive COst MOdel (COCOMO) was developed in 1981 by Barry Boehm and was published along with a data set containing 63 projects [4]. At that time, Boehm published the COCOMO model in three versions (all are considered COCOMO 81 hereafter). *Basic COCOMO* is a relatively simple model "good for quick, early, rough order of magnitude estimates of software costs" with a limited accuracy [4]. Basic COCOMO applies to three modes of software projects, *organic*, *semidetached*, and *embedded*. In the organic mode "relatively small software teams develop software in a highly familiar, in-house environment," whereas in the embedded mode, "project should operate within tight constraints (strongly coupled complex of hardware, software, regulations, and operational procedures) [4]." The semidetached mode is "an intermediate stage between the organic and embedded modes [4]." Basic COCOMO only uses the lines of code (or delivered source instructions, DSI) in its estimates. Due to the limited accuracy of Basic COCOMO, being "within a factor of 1.3 of the actuals only 29% of the time, and within a factor of 2 of the actuals only 60% of the time [4]," as well as other limitations, this study does not focus on Basic COCOMO. Instead, *Intermediate COCOMO* is used in this research.

The core intuition behind COCOMO is that, as a program grows in size, the development effort grows exponentially. This is the case with either models. Similar to Basic COCOMO, Intermediate COCOMO also supports the three software modes. The constants for these modes, which are different from Basic COCOMO, are provided in Table 2.1. The equation for estimating effort in these modes [4] is

$$effort = a * \left(KLOC^{b}\right) \tag{2.1}$$

where *a* and *b* are the constants in Table 2.1 and *KLOC* is the thousand lines of code (or delivered source instructions). Effort is measured in man-month, which consists of 152 hours of working time and includes those activities related to the development or managing a project.

According to Boehm, "Intermediate COCOMO incorporates an additional 15 predictor vari-

Development Mode	a	b
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.20

Table 2.1: Intermediate COCOMO 81 development modes.

	Cost Driver	Description
Upper (increase these to decrease effort)	ACAP	Analyst capability
	PCAP	Programmer capability
	AEXP	Applications experience
	MODP	Use of modern programming practices
	TOOL	Use of software tools
	VEXP	Virtual machine experience
	LEXP	Programming language experience
Middle	SCED	Required development schedule
Lower (increase these to increase effort)	DATA	Data base size
	TURN	Computer turnaround time
	VIRT	Virtual machine volatility
	STOR	Main storage constraint
	TIME	Execution time constraint
	RELY	Required software reliability
	CPLX	Product complexity

Table 2.2: COCOMO 81 cost drivers.

ables which account for much of the software project cost variation left unexplained by Basic COCOMO [4]." This extension to Basic COCOMO provides a better accuracy, where "the estimates are within 20% of the project actuals 68% of the time [4]." The 15 factors, also called *cost drivers*, are provided in Table 2.2.

As a result, equation 2.1 is extended [4] to include these cost drivers.

$$effort = a * \left(KLOC^{b}\right) * \left(\prod_{j} EM_{j}\right)$$

Here, $KLOC$ is the thousand lines of code, EM_j is one of the cost drivers from Table 2.2 with precise values from Table 2.3, and a and b are the parameters tuned by Boehm's local calibration

	Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Upper (increase these to decrease effort)	ACAP	1.46	1.19	1.00	0.86	0.71	
	PCAP	1.42	1.17	1.00	0.86	0.70	
	AEXP	1.29	1.13	1.00	0.91	0.82	
	MODP	1.2	1.10	1.00	0.91	0.82	
	TOOL	1.24	1.10	1.00	0.91	0.83	
	VEXP	1.21	1.10	1.00	0.90		
	LEXP	1.14	1.07	1.00	0.95		
Middle	SCED	1.23	1.08	1.00	1.04	1.10	
Lower (increase these to increase effort)	DATA		0.94	1.00	1.08	1.16	
	TURN		0.87	1.00	1.07	1.15	
	VIRT		0.87	1.00	1.15	1.30	
	STOR			1.00	1.06	1.21	1.56
	TIME			1.00	1.11	1.30	1.66
	RELY	0.75	0.88	1.00	1.15	1.40	
	CPLX	0.70	0.85	1.00	1.15	1.30	1.65

Table 2.3: The precise COCOMO 81 effort multiplier values.

procedure (further discussed in Section 3.5.5). Cost drivers may increase or decrease the effort. This can be seen with labels upper or lower in Tables 2.2 and 2.3.

The third and last version of COCOMO 81 is the *Detailed COCOMO*, which provides “a set of phase-sensitive effort multipliers for each cost driver attribute” and takes into account an assessment of the impact of each cost driver in different levels of development [4]. Detailed COCOMO is rather complex and was not used in this study.

COCOMO 81 has two public data sets (*COC*81 and *NASA*93, discussed in Section 3.1.1) and several researchers have published baseline results based on COCOMO [13,46]. According to Jorgensen and Shepperd, where they reviewed effort estimation studies from 1989 to 2004, “roughly half of all estimation papers try to build, improve or compare with regression model-based estimation methods” with COCOMO being one of the most common parametric estimation model in this dominant category of research [30].

Boehm later updated the COCOMO 81 model to COCOMO II in 2000 [7]. COCOMO II adds five *scale factors* that exponentially influence effort. However, COCOMO II is not used in this study since there is no publicly available data set for it. Hence, COCOMO II is not further discussed.

2.2.2 Other Models

There are several other software models. This study follows previous studies in the literature that uses COCOMO and hence, it does not explore these models. Furthermore, there is either no publicly available data or few baseline results for some of these models, making it hard to compare the results of this study with others. However, as Section 4.1 mentions, these models can be studied in future works upon the availability of data. Boehm suggests this as a good practice as well [5].

Some of these models and approaches with references are listed below:

- CHECKPOINT: is a commercial model [19].
- ESTIMACS: is a commercial model [33].
- PRICE-S [55] is privately owned [19].
- REVIC: Revised Enhanced Intermediate Version of COCOMO, is privately owned by the government [19].
- SAGE: is a commercial model [19].
- SASET: Software Architecture Sizing and Estimating Tool, is privately owned by the government [19].
- SEER-SEM: System Evaluation and Estimation of Resources — Software Estimating Model [26] is privately owned [19].
- SLIM: Software Lifecycle Management [59] is privately owned [19].
- SoftCost: is a commercial model [19].

2.3 Column Pruning

Column pruning, also known as feature subset selection [21] and variable subset selection [49], is a pre-processing method to select a subset of columns of a data set, while removing the others, which could reduce the noise in the data. Reducing the noise can improve the accuracy of the trained model, hence improving the performance of an effort estimation method. Chen et al. [12] as well as other researchers [2, 25, 34] have reported this as well.

The reduction of noise can be more clearly explained in mathematical notations. In a linear model with constants β_i that inputs features f_i from a set of features F to predict for y,

$$y = \beta_0 + \beta_1 \cdot f_1 + \beta_2 \cdot f_2 + \beta_3 \cdot f_3 + \cdots$$

the variance of y is some function of the variances in f_i. If the set F contains noise, then random variations in f_i can increase the uncertainty of y. Column pruning methods decrease the number of features f_i, thus increasing the stability of y's predictions. That is, the fewer the features, the more restrained are the model predictions. Taken to an extreme, column pruning can reduce y's variance to zero by pruning the above equation back to $y = \beta_0$, but increases model error since the equation $y = \beta_0$ will ignore all project data when generating estimates. Hence, intelligent column pruners experiment with some proposed subsets $F' \subseteq F$ before changing that set.

The removed columns are not necessarily useless. They are, however, not suitable for the model being generated from the data. Therefore, the same column may not be removed if another column pruner was used. The inclusion of such columns happens during the data collection process when several features are included in the data for the possibility of being used in the future. Column pruners may also remove redundant columns if they exist, which can improve the efficiency of the learning algorithm and produce smaller theories and models [11].

Column pruning is generally used in machine learning, and is often referred to as attribute selection. According to Hall and Holmes [21], there are several attribute selection methods, which are often categorized as either *filter*s or *wrapper*s. The main difference between the two is in their evaluation method. A wrapper uses a target learner to evaluate an attribute while a filter uses general characteristics of the data and does not use the target learner in evaluating the attributes [21]. There are other classifications of attribute selection methods as well. Ranking attributes individually or in subsets and ranking them based on their class type (being numeric or discrete) are among such classifications [21].

Hall and Holmes [21] compared six attribute selection methods, including information gain attribute ranking (a simple and fast method that uses entropy and information gain measures), Relief (an instance-based method with its extension, ReliefF, to handle noise and multiclass data sets), Principal Component (an statistical technique that can reduce the dimensionality of data using transformations), CFS (Correlation-based Feature Selection that evaluates subsets rather than individual attributes using some heuristic), consistency-based subset evaluation (that uses class consistency as an evaluation metric), and the Wrapper attribute selection method. They benchmarked their results across 15 UCI data sets [3] and against C4.5 [61] and naive Bayes [18]. Their results show that attribute selection can improve the performance of common learning algorithms and that there is no single best attribute selection method for all cases. However, they show that [21] Wrapper is the most accurate attribute selection method (when speed is not an issue).

Hall and Holmes' study has led this study to explore different column pruning methods, including Wrapper and a variation of Wrapper called Local Wrapper developed by Chen et al. [12, 46], and also simple, near-linear-time column pruning methods developed by Baker [2]. These column pruning methods are described in Section 3.5.3.

Column pruning methods can search for the best attributes using *forward selection* or *backward elimination* of the attributes. In forward selection, the column pruner starts with an empty subset and adds attributes to the list as they are evaluated. In backward elimination, the column pruner starts with the the set of all attributes and removes them one at a time as they are evaluated. These two approaches, as well as the combination of both [2], are applied in several of the column pruners studied in this study.

2.4 Row Pruning

Row pruning, also known as stratification, is used as a pre-processor in effort estimation. For a given project, row pruning finds similar projects and uses them in estimating the effort for the given project. This is especially helpful when calibrating an effort estimation method since similar projects have less variation. Row pruning is applicable to several software models, including COCOMO 81. Ferens and Christensen stratified data for all of the nine software models they used (REVIC, SASET, PRICE-S, SEER-SEM, SLIM, SoftCost, CHECKPOINT, COCOMO II, and SAGE) before calibration [19].

Researchers have studied row pruning in effort estimation extensively [12, 40, 45–47, 67, 68]. The problem of *outliers* have been reported in [43]. Outliers are projects that exist in a data set used for calibration of an effort estimation method and negatively affect the produced model. Row pruning is an ideal method for identifying and removing such outliers. (An earlier and smaller version of this study did not report row pruning as an effective method. Column pruning methods usually outperformed row pruning methods [25]. However, this is considered to be a problem with the evaluation methods in that study. This study reports row pruning as an effective method, as it can be seen in Section 3.6.)

There are many other studies that predict row pruning can improve the performance of methods used in effort estimation in general. Menzies et al. reported large improvements in terms of PRED(N), defined in Section 2.6.1, when using stratified data instead of unstratified data [45]. In another study, they report that pruning in general is most important for small datasets [12]. This may seems reasonable since an outlier or an irrelevant attribute can more negatively affect a model when the size of the data set is smaller. Such outliers can be removed using row pruning and the irrelevant attributes can be removed using column pruning. (However, this study has not found any relation between the data set size and the effectiveness of row or column pruning methods. Their conclusion could be related to the wrong evaluation criteria they used.)

This study only comments on the effectiveness of row pruning for effort estimation in COCOMO 81 models. However, in a study using COCOMO II, SEER-SEM, and PRICE-S models, Lum et al. predicted that adjusting the models for the local environment should improve the performance of these models [40]. Such adjustments can be done using row pruning.

There are also studies that report negative results from certain row pruning methods. In a more recent study using stratification, Menzies et al. reported negative results for the *NASA*93 data set [46]. This data set is further explained in Section 3.1.1. (This is not consistent with the results of this study on stratification. The reason could be the wrong evaluation criteria they used.)

Stratification is not limited to software effort estimation. Cost estimation in general is a wide research area and researchers have tried to use similar methods, such as stratification, for cost estimation in many other fields. Chen and Huang studied stratification for Monte Carlo production cost simulation [23], which is a probabilistic method. Their study compares a proposed stratification method with conventional stratification methods and reported better variance reduction in estimates using their proposed methods.

2.4.1 Manual and Automatic Row Pruning

Previous studies such as [12,46] mostly focused on manual row pruning (also discussed in Section 3.2.1). However, there exists an automatic approach as well, which was developed in this study and is discussed in Section 3.2. There are similarities between automatic row pruning methods and nearest neighbor methods, which are estimation by analogy methods. As discussed before, estimation by analogy is a many-data method and is therefore suitable for selecting similar instances from the historical data to find an estimate for a new project. In general, case-based reasoning methods, which encompasses estimation by analogy, can be used as automatic row pruning methods. This is discussed in the next section.

2.4.2 Case-Based Reasoning and Row Pruning

As mentioned in Section 2.1.4, case-based reasoning (CBR) is a more general approach for finding similarities between past projects and a current project. While row pruning methods use a similar approach to CBR, they may differ in the way CBR trains a model. CBR finds solutions to similar existing problems to generate a solution to a current problem. In training, a row pruning method may function similar to a rule induction algorithm (as discussed before), in that it trains a model without any knowledge of the current problem. However, in order to train the best model possible, a train instance can simulate the role of the current problem in training the model, similar to CBR. It is clear that such a train instance may or may not be similar to the current problem. Hence, the resulting model may not perform well. There are similarities between estimation by analogy (discussed in Section 2.1.6) as a type of CBR and the row pruning methods used in this study (discussed in Section 3.2).

Since CBR is model free [67], using CBR methods (such as the nearest neighbor algorithm, discussed in Section 3.2.2) in effort estimation can reduce the need for a certain model such as COCOMO. The reason is that, nearest neighbor algorithms make few or no assumptions about the domain. Therefore, they can be applied to a variety of data sets based on different effort estimation models. However, this can also be considered a drawback of this algorithm since ignoring important domain knowledge and searching for similar instances "blindly" can later lower the performance of the learned model or increase the error in the estimate. (This is confirmed by the current study as it can be seen in Section 3.6. There is a solution to this problem and that is to augment the nearest neighbor algorithm with another method that takes into account the domain assumptions. The row pruning method developed in this study follows this approach. This is further discussed in Section 3.2.3.)

2.5 Learners

Learners are essentially machine learning tools that can find a pattern in the data or generate a model, among many other application for them. In effort estimation, learners are used to learn a model from the historical data, which can later be applied to new projects. The learned theory is used to find an estimate for those new projects. This study generally applies the name learner to any effort estimation method that can find an estimate. There may be other classifications for learners as well. For example, the nearest neighbor algorithm discussed before (or other CBR methods) do not generally learn a model. They use the historical data to find an estimate for a new project without generating a model that can be applied to a second project, meaning that, there is no actual learning involved. (As a result, when the nearest neighbor is analyzed in Section 3.6, it is shown without a learner and only as a row pruner.)

This study focuses on regression methods such as linear regression, local calibration [4], and model trees [60] as learners. Linear regression and model trees can be found in Weka, Waikato Environment for Knowledge Analysis, along with numerous other machine learning tools [76]. These learners are discussed in detail in Section 3.5.5.

In general, different machine learning methods can be combined in different orders to create different methods. Effort estimation methods can also be combined this way. The goal is to create a better estimation method using the aggregate performance of such methods. Clearly, not all combinations will result in a more powerful method due to the conflicting nature of some methods.

Theoretically, the *stacking* or combining of *n* methods can be done in *n*! ways. A *meta-learner* is used in stacking to learn which methods are reliable and find the best way to stack different methods [76]. This process becomes harder as more methods have to be considered. The number of available methods is large and grows larger with new data mining toolkits appearing more frequently. Weka, the R Project (available online at `http://www.r-project.org/`), and Orange (available online at `http://www.ailab.si/orange/`) are among such toolkits. Therefore, such a large number of methods can potentially create thousands of combinations, which must be evaluated and pruned before the search space gets filled with several over-elaborate methods. (This study compares 312 combinations and prunes the majority of them.)

Bagging and *boosting* are among other meta-learning algorithms [76]. Their application in effort estimation was recently investigated by Baker [2] and no significant improvement in model accuracy was reported. This study does not explore these methods.

2.6 Evaluation

Evaluation of effort estimation methods is probably as important as finding such methods. Without a proper evaluation of these methods, no decision can be made about their relative performance. Hence, effort put into developing them may have well been wasted. Usually the comparison of effort estimation methods is done through some statistical test that compares their error distributions [76]. It is important to choose the right statistical test for a comparison. This section describes the benefits and limitations of two different categories of tests.

First, it is helpful to understand different ways of collecting error distributions. In general, error is defined as the difference between the actual and expected behavior of a certain object. In effort estimation, error of a method is measured as a function of the actual effort needed and the prediction (or estimate) of effort provided by that method. Here are some of the most used error definitions in effort estimation:

- *Absolute Residual* (AR) is defined as

$$|actual - prediction|$$

  which is one of the simplest error definitions and is the absolute value (magnitude) of the difference between actual and predicted efforts. Another terminology, also used in this study, for AR is *Magnitude of Error*, ME.

- *Relative Error* (RE) is defined as

  $$\frac{actual - prediction}{actual}$$

  and has both negative and positive error values possible.

- *Magnitude of Relative Error* (MRE) is defined as

  $$\frac{|actual - prediction|}{actual}$$

  which is simply the magnitude of RE.

- *Magnitude of Error Relative to the estimate* (MER) is defined as

  $$\frac{|actual - prediction|}{prediction}$$

  and differs from MRE in the fact that it measures the error relative to the predicted value. MER was proposed by Kitchenham et al. [36].

- *Balanced Relative Error* (BRE) is defined as

  $$\begin{cases} \frac{prediction-actual}{actual} & \text{if } prediction - actual \geq 0 \\ \frac{prediction-actual}{prediction} & \text{if } prediction - actual < 0 \end{cases}$$

  and was suggested by Miyazaki et al. [50].

- *Inverted Balanced Relative Error* (IBRE) is defined as

$$\begin{cases} \frac{prediction-actual}{actual} & \text{if } prediction - actual < 0 \\ \frac{prediction-actual}{prediction} & \text{if } prediction - actual \geq 0 \end{cases}$$

  and was also suggested by Miyazaki et al. [50].

Some of these error definitions such as AR, MER, and MRE are used in many studies [20, 36, 54, 66, 67]. On the other hand, BRE and IBRE have not been used in many studies and the study by Foss et al. does not report the tests based on these any better than the ones based on MER or MRE [20]. Therefore, this study uses AR, MER, and MRE as the *evaluation criteria* in the analysis of results of different effort estimation methods.

Many statistical tests only specify whether the distributions are identical (equal) or not. This is also known as verifying the *null hypothesis* that the two samples are identical. A null hypothesis is a hypothesis compared with an alternative hypothesis using statistical tests. This type of tests is frequently used and several well-known tests such as *Student's t-test* are among the tests in this category.

Other tests can offer comparisons where one distribution can be considered better or worse than another. This will offer conclusive results about the performance of test subjects represented with these distributions. Hence, these tests may be preferred over the tests that only verify the null hypothesis. However, not many statistical tests can provide such a comparison. Usually, these tests are created based on the needs of the study by augmenting a test that verifies the null hypothesis with a simple test, such as the median or mean comparisons. In this case, if the null hypothesis is rejected, meaning that the distributions are not identical, the medians (or means) of each distribution are compared to find which one performed better.

In general, statistical tests are categorized into two groups of *parametric* and *non-parametric* tests. These are defined in the following sections.

2.6.1 Parametric Tests

Parametric tests assume parameterized distributions in the data. The most common parameterized distribution is the *normal* or *Gaussian* distribution. Parametric tests such as the t-test and ANOVA (analysis of variance) assume a normal underlying distribution. Student's t-test is one of the most popular parametric tests. Using its different versions, it can be applied to two independent distributions (where each distribution contains a random instance, which may or may not have a relationship with another instance in the other distribution) or to two paired distributions (where each instance of one distribution has a unique relationship to an instance of the other distribution). It can be used to test whether two error distributions (based on MRE for example) are statistically the same or not [11, 12, 46]. However, there are several limitations to a parametric test such as the t-test.

The main limitation of a parametric test is that it assumes an underlying distribution when such a distribution may or may not exist. For example, several previous studies [12, 25, 45, 46] uses t-tests to compare the error distributions. This study repeated the experiments in [25, 46] and as Figure 3.1 shows, the underlying distributions are not normal. Hence, there is a need for checking the underlying distribution before proceeding with parametric tests, which are frequently used in the literature. Such verifications may certainly fail in different cases and hence, there is a need for tests that do not assume an underlying distribution. Such tests belong to the category of non-parametric tests, which is described in the next section.

Another limitation of a parametric test such as the t-test is that they are usually susceptible to outliers (defined before). Outliers have large deviations comparing to other instances in the distribution and greatly affect the mean and standard deviation calculations used frequently in parametric tests. As Menzies et al. mention in their study [46], the presence of large deviations makes it harder to assess the performance of different methods. Therefore, comparisons will show instability and evaluation of effort estimation methods becomes pointless. The instability problem is discussed further in Section 2.7. One way to avoid the problem of outliers is to use rankings of error values, instead of the error values, to compare methods. This is the approach in some non-parametric tests, as described in the next section.

Student's t-test is not the only parametric test used. Most of the studies mentioned before also used mean and standard deviations of the error estimates, such as MRE, in their evaluation methods. Although mean MRE is more frequently used in the literature than other mean values, it is neither the best choice nor it is suitable in every case. In a study by Foss et al. [20], mean MRE (MMRE), mean MER (MMER), mean BRE (MBRE), mean IBRE (MIBRE), and median MRE (MdMRE) were compared to SD, RSD, and LSD, which are defined below. (Unlike all other tests in their study, MdMRE is not a parametric test.) According to the results of their study, although "MMRE has for many years been, and still is, the de facto criterion to select between competing prediction models in software engineering," it is "an unreliable criterion" and therefore, cannot be relied on in comparison of effort estimation methods [20]. They suggest MMER to be preferred over MMRE and conclude that MMRE, MMER, MBRE, MIBRE, and MdMRE are "substantially more unreliable as evaluation criteria than SD, RSD, and LSD [20]." Even so, they consider none of these as universal solutions for comparing effort estimation methods due to either a flaw or a limitation in these criteria.

Standard Deviation (SD) is defined as

$$\sqrt{\frac{\sum(actual_i - prediction_i)^2}{n-1}}$$

where n is the size of the distribution and $actual_i$ and $prediction_i$ are the corresponding values for each instance in the distribution.

Relative Standard Deviation (RSD) is defined as

$$\sqrt{\frac{\sum(\frac{actual_i - prediction_i}{x_i})^2}{n-1}}$$

where x_i is a certain variable in the data such as function points, FP [20].

Logarithmic Standard Deviation (LSD) is defined as

$$\sqrt{\frac{\sum(e_i - (-\frac{s^2}{2}))^2}{n-1}}$$

where e_i is define as

$$\ln actual_i - \ln prediction_i$$

and s^2 is the variance of e_i [20].

Another popular parametric test using MRE is PRED(N) defined as

$$\frac{1}{T}\sum_{i=1}^{T}\begin{cases} 1 & \text{if } MRE_i \leq N \\ 0 & \text{if } MRE_i > N \end{cases}$$

and is usually used with $N = 0.25$ and $N = 0.30$. It is also described in terms of percentages such as PRED(30). PRED(N) is used in many studies where MMRE and other MRE-based methods are used [10, 45, 46, 69]. However, Foss et al. "do not believe that reporting several measures that are all based on MRE would improve matters" in comparing methods [20]. Menzies et al. [46] suggest using a combination of parametric tests, which they call *rejection rules*. This is further discussed in Section 3.3.2.

2.6.2 Non-Parametric Tests

Non-parametric statistics includes a large group of tests that do not assume an underlying distribution and are hence considered *distribution free*. This makes non-parametric tests a suitable candidate for effort estimation studies since, unlike parametric tests, they do not require verification of the underlying distribution. Moreover, if such a verification fails, non-parametric tests become the only possible choice. In general, due to fewer assumptions they make, non-parametric tests are considered statistically more robust (meaning that violations of their assumptions do not affect them unpredictably and they still perform well).

The application of non-parametric tests in effort estimation studies [10, 17, 20, 31, 36] have not been as frequent as the parametric ones. Usually, simple non-parametric tests are used. For example, Foss et al. studied median MRE [20], which can be considered a non-parametric test since it does not need to have any specific distribution. Other researchers have used median values as well. However, similar to the results from [20], no conclusive results were offered as to which evaluation method performed well for general use. Demsar [17] suggests using non-parametric tests, which are mentioned to be "simple, yet safe and robust." Demsar's study analyzes other papers and compares different evaluation methods on classifiers' performance. In order to compare two classifiers, Demsar uses methods such as averaging over data sets, paired t-tests, Wilcoxon signed-rank test [74], and finally counts of wins, losses, and ties based on a sign test (also suggested by [8]). The non-parametric Wilcoxon signed-rank test is considered an alternative to the parametric paired t-test [10, 17]. When comparing multiple classifiers, Demsar uses ANOVA and Friedman test, which was augmented with other tests in case the null hypothesis was rejected. Friedman test is a non-parametric equivalent of the parametric test ANOVA.

Demsar's results show that the parametric test ANOVA is no better than the non-parametric Friedman test. More generally, using empirical results, Demsar suggests using the two non-parametric tests, Wilcoxon signed-rank and Friedman tests. Although the results of Demsar's study clearly suggest non-parametric tests, there are better non-parametric tests that can be used in a wider range of experiments. More specifically, Friedman test requires other additional tests and Wilcoxon signed-rank test can only be applied where the two samples are related or are two replications of the same experiment on the same sample.

Another non-parametric test that verifies the null hypothesis is the Mann-Whitney U test [42], which is discussed in detail in Section 3.4.2. Although the Mann-Whitney test can only verify the null hypothesis, a simple comparison of the two medians of the ranks for each distribution, generated during the procedure, will determine the preferred distribution. This test shows a great amount of stability as shown in Section 3.6.

There are other alternatives for non-parametric tests as well. Kitchenham et al. [36] suggest using *boxplot*s as alternatives to parametric tests such as MMRE and PRED(N). According to their study, boxplots show the median values as the central location for the distribution and show two tails for each boxplot. A boxplot shows the spread of the distribution using the height of the box and the skewness of the distribution using the position of the median and the length of boxplot tails. Using the residuals, $prediction - actual$, is suggested as one of the possible variables for boxplots.

Menzies et al. have a similar approach using *quartile charts* [48]. A quartile chart is generated by sorting the distribution and dividing the results into four quartiles. The median is represented with a dot and the upper and lower quartiles are represented with vertical lines.

Although boxplots and quartile charts are non-parametric tests that summarize the data, they use visual representations. Analyzing visual representations in studies where a large number of methods are used across different data sets requires a great amount of time and accuracy and therefore, offering a summary of the results becomes a cumbersome task. As a result, this study does not focus on boxplots or quartile charts for evaluating its large number of methods.

2.6.3 Evaluation Bias

There are numerous evaluation methods and inherently, every effort estimation study ignores some evaluation method. Therefore, every study is biased toward a certain group of evaluation methods. In general, evaluation bias cannot be avoided and without it no major study can find any conclusive results, since each evaluation method can suggest a different conclusion. Nonetheless, its extent can be reduced to a minimum such that the choice of evaluation methods does not affect the results as much while making it possible to find stable conclusions.

Evaluation bias can exist in effort estimation studies in different forms. One of the most common evaluation biases is the way that an evaluation method behaves across different data sets. In order to avoid several underlying characteristics of a data set that may unpredictably affect the conclusions, Shepperd and Kadoda's study uses simulated data to control these characteristics [67]. Their study shows that as the characteristics of the data sets change, the conclusions made using the same methods also change. However, none of the methods studied could be considered better than others in the comparisons made using several evaluation methods across the simulated data sets with different characteristics. Hence, different data sets can affect the evaluation of the same methods in different ways, causing evaluation bias in the study.

Another place that evaluation bias is visible is when comparing different studies using the same methods. Menzies et al. reported that their rejection rules, a type of evaluation method they created, reported that stratification was useful [12]. A later study [25] using the same rejection rules and experimental approach contradicted the previous results, showing that stratification was not useful. This shows the amount of instability some evaluation methods can create when an experiment is repeated.

Finally, evaluation bias can be caused by the underlying criteria used in the evaluation method. For two underlying criteria of MRE and MER used in a simple mean test, Foss et al. argue that mean MER is a better test than mean MRE [20]. Although this test is very simple, it shows how much different criteria can affect a test. More sophisticated tests are usually more affected by different underlying evaluation criteria since there are many factors involved and changing one could have an adverse effect on the others.

This study aims at reducing the evaluation bias in comparing effort estimation methods by using a better test, several evaluation criteria, different data sets and subset of those data sets, and several randomized runs of the same experiment to produce different results.

2.7 Instability

Stability is a defining factor in accepting the conclusions of a study and without it, it is impossible to compare studies, approaches, or methods. The effort estimation literature has reported many instabilities in the results and as a result, no clear conclusion has been accepted widely as to which method or methods perform the best. For example, Shepperd and Kadoda [67] compared different methods and reported no stable conclusions despite their many efforts.

Recently, some researchers have realized this problem and attempted at analyzing its sources [20, 36, 54, 67]. Among these studies, some were pessimistic about finding a possible solution to the problem while a few suggested general approaches without a detailed plan. Foss et al. [20] compared several evaluation methods and reported no clear conclusion about the preferred method as a universal evaluation method. As a result, they state that, "it probably is futile to search for the Holy Grail: a single, simple-to-use, universal goodness-of-fit kind of metric, which can be be applied with ease to compare" different methods [20].

Myrtveit et al. [54] consider the problem of instability a "poorly understood" problem and suggest that the research procedure in effort estimation studies (using a single data sample, an accuracy indicator, and cross-validation) is the source of the problem due to its unreliability. As a result, they "cast some doubt on the conclusions of any study of competing software prediction models that used this research procedure as a basis of model comparison [54]." Their solution is a rather general solution, which suggests developing more reliable research procedures.

Kitchenham et al. [36] consider it essential to understand how to make comparison between competing models and mention that some of the different accuracy indicators used by different researchers generated contradictory results. They continue, "without understanding what the various indicators are describing, meaningful comparison is not possible. Furthermore, if we cannot make meaningful comparisons we cannot make progress [36]." Once again, only a general solution is given for the problem of instability in comparing methods. Shepperd and Kadoda's study [67] suggests exercising caution when comparing different approaches along with few general guidelines and methods.

Although none of the above studies suggested a clear solution to the problem of instability in comparing effort estimation methods, this does not mean that such a solution does not exist. As a matter of fact, this study reports stability in comparing a wide range of effort estimation methods across different data sets, evaluation criteria, and randomized runs. This might be the first study that reports such stability and it is expected that future studies will find similar stabilities as well.

2.8 Summary of Current Problems

There are many problems that an effort estimation study may face. Four of the more prominent problems are summarized here.

- Effort estimation studies lack stability in their conclusions in general. Recently, researchers attempted at finding the source of such instabilities. However, no clear approach is suggested to reduce or diminish instability.

- Lack of publicly available data sets has affected effort estimation studies greatly since their suggested methods cannot be tested thoroughly. Moreover, these data sets may not be suitable for every study since they were collected using certain models. Unless a researcher can access private or commercial data, which makes the experiments unrepeatable for others, the study has to use simulated data. Simulated data has its own limitations and cannot always represent the real-world examples.

- There are too many effort estimation methods and without a proper pruning of this list, researchers will not be able to compare them conclusively. Hence, little progress can be made in finding better methods.

- The evaluation approaches used in effort estimation studies need to be revised and better approaches, accepted by the majority of researchers, have to be applied in order to reduce the instability present in the current studies.

This research applies better evaluation criteria and as a result, offers solutions for the instability problem. It also prunes many effort estimation methods and provides a framework for continuing this pruning. Although in this study the lack of data is offset by using different subsets of data sets, this problem requires the help of the software engineering industry by releasing their data, which will ultimately benefit them by providing them with better effort estimation methodologies.

Chapter 3

Laboratory Studies

This chapter describes the laboratory studies performed throughout this research. In this work, COSEEKMO served as the framework in which many types of experiments became possible. As a result of these studies, several tools such as Locomo were developed. Also, an existing systematic error in the evaluation of methods became clear, which resulted in using a new evaluation method called the Mann-Whitney U test. Finally, many of the methods developed by other researchers such as Shepperd's nearest neighbor method [67] and Baker's column pruning method [2] were incorporated in COSEEKMO.

The experimental design is described in Section 3.1, including the data used in this study, the experimental method, and a general description of the evaluation of the results. This study focuses on column and row pruning among other methods for effort estimation. Column pruning is discusses in detail in [2] and is mentioned in the methods used in COSEEKMO in Section 3.5.3. On the other hand, part of this study is mainly about different row pruning methods. These methods are discussed in Section 3.2.

Another part of this study is designated to the evaluation methods used to compare effort estimation methods. Sections 3.3 and 3.4 explain several different parametric and non-parametric tests used respectively. Section 3.4.2 explains the Mann-Whitney U test, a non-parametric test, in detail.

Section 3.5 categorizes all different methods in COSEEKMO and space of combined methods possible in COSEEKMO. Such combinations are used in this study to find the group of best methods suitable for effort estimation. (A manual for COSEEKMO is provided in Appendix A.) Section 3.6 provides the results and discusses the findings based on these results.

3.1 Experimental Design

All the experiments in this work were done using the COSEEKMO tool. This study focus on effort estimation methods based upon Boehm's 1981 version of the COCOMO model [4], also called COCOMO 81, since there is publicly available data sets in this model's format. Boehm's updated version of COCOMO, called COCOMO II [7], can also be supported in the methods present in COSEEKMO as it is shown for one such method called Cocomin [2]. However, since there is no publicly available data set in the COCOMO II format, this study does not explore this model. The COCOMO model is further explained in Section 2.2.1.

3.1.1 Data

The data used in this study was from two data sets:

- *COC*81 containing 63 records in the COCOMO 81 format [4, p496-497] and
- *NASA*93 containing 93 NASA records in the COCOMO 81 format.

Taken together, these two data sets are the largest COCOMO-style data source in the public domain (for reasons of corporate confidentiality, access to Boehm's COCOMO II data set is highly restricted). Boehm used the 63 projects present in *COC*81 in his study to calibrate the COCOMO 81 model. *NASA*93 was originally collected to create a NASA-tuned version of COCOMO, funded by the Space Station Freedom Program. It contains data from six NASA centers including the Jet

Propulsion Laboratory. Menzies et al. report that there is a large deviation seen in the *NASA*93 data [46]. According to them, this was caused by the presence of a wide variety of projects rather than poor data collection. They explain the process of data collection in detail and believe that "*NASA*93 was collected using methods equal to or better than standard industrial practice [46]." However, this work shows that the reason for the large deviations in *NASA*93 as well *COC*81 is the evaluation method used, which was based on parametric assumptions. These assumptions are shown not to hold. Therefore, their belief that equal or larger deviation can be seen in industrial data based on what they observed in *NASA*93 does not hold.

In this study, effort estimators were built using all or some part of data from *COC*81 and *NASA*93 data sets. These two data sets contain several different subsets. The name of these subsets and the number of subsets used (in parenthesis) are:

- All (2): selects all records from a particular source.
- Category (2): *NASA*93 designation selecting the type of project such as avionics.
- Center (2): *NASA*93 designation selecting records relating to where the software was built.
- Fg (1): *NASA*93 designation selecting either "*f*" (flight) or "*g*" (ground) software.
- Kind (2): *COC*81 designation selecting records relating to the development platform such as max for mainframe.
- Lang (2): *COC*81 designation selecting records about different development languages such as ftn for FORTRAN.
- Mode (4): designation selecting records relating to the COCOMO 81 development mode such as semi-detached, embedded, or organic.
- Project (2): *NASA*93 designation selecting records relating to the name of the project.

Number	Data Set	Name	Size
1	*COC*81	all	63
2	*COC*81	kind-max	31
3	*COC*81	kind-min	21
4	*COC*81	lang-ftn	24
5	*COC*81	lang-mol	20
6	*COC*81	mode-e	28
7	*COC*81	mode-org	23
8	*NASA*93	all	93
9	*NASA*93	category-avionicsmonitoring	30
10	*NASA*93	category-missionplanning	20
11	*NASA*93	center-2	37
12	*NASA*93	center-5	39
13	*NASA*93	fg-g	80
14	*NASA*93	mode-embedded	21
15	*NASA*93	mode-semidetached	69
16	*NASA*93	project-gro	23
17	*NASA*93	project-X	38
18	*NASA*93	year-1975	37
19	*NASA*93	year-1980	38

Table 3.1: Names and sizes of subsets of *COC*81 and *NASA*93.

- Year (2): *NASA*93 term that selects the development years, grouped into units of five. For example, years 1970, 1971, 1972, 1973, and 1974 are labeled "1970".

Overall, *COC*81 and *NASA*93 data sets contain 57 subsets (including the super sets). However, only 19 of these subsets were used in this study. Their names and sizes are provided in Table 3.1. The reason is that the rest of the subsets contain fewer than 20 projects (with 25 of them containing fewer than 10 projects). As described in [46], due to choosing a test set size of 10, there will not be enough projects to train a model in the subsets that contain fewer than 20 projects.

The subsets of each data set can overlap with each other since a project may be categorized in different ways (for example year and center may share projects). Therefore, different projects may belong to more than one subset of each data set. These overlaps for *COC*81 and *NASA*93 data sets are provided separately in tables 3.2 and 3.3 respectively. It should be noted that the subset *all* is

Subset	2	3	4	5	6	7
1	31/63	21/63	24/63	20/63	28/63	23/63
2		0/52	16/39	2/49	10/49	15/39
3			6/39	14/27	16/33	4/40
4				0/44	7/45	12/35
5					13/35	4/39
6						0/51

Table 3.2: Overlap of subsets of *COC*81 shown as intersection/union for each pair of subsets. Subset numbers match the ones in Table 3.1.

Subset	9	10	11	12	13	14	15	16	17	18	19
8	30/93	20/93	37/93	39/93	80/93	21/93	69/93	23/93	38/93	37/93	38/93
9		0/50	13/54	17/52	30/80	3/48	24/75	4/49	17/51	20/47	5/63
10			1/56	7/52	20/80	2/39	18/71	1/42	7/51	3/54	15/43
11				0/76	32/85	5/53	32/74	23/37	0/75	10/64	13/62
12					33/86	13/47	23/85	0/62	38/39	23/53	14/63
13						8/93	69/80	20/83	33/85	31/86	31/87
14							0/90	3/41	12/47	12/46	9/50
15								20/72	23/84	25/81	27/80
16									0/61	0/60	9/52
17										23/52	13/63
18											0/75

Table 3.3: Overlap of subsets of *NASA*93 shown as intersection/union for each pair of subsets. Subset numbers match the ones in Table 3.1.

the super set for each data set and therefore will overlap with every other subset. In each data set, the overlap is shown as $\frac{|A \cap B|}{|A \cup B|}$ for all pairs of subsets A and B.

3.1.2 Experimental Method

This work is based on cross-validation. COSEEKMO uses cross-validation to partition the data into train and test sets. COSEEKMO does this is two different ways in order to study the effects of stratification (discussed later in Section 3.2) suggested by Boehm [7]. Each subset (of *COC*81 or *NASA*93) is partitioned into a test set and a train set. Also, all the instances in the superset (of the subset) that are not present in the test set are stored in another train set. The first train set is used when studying the effect of stratified data and the second train set is used when studying the effects of non-stratified data. In each run, each train set is used individually to train a model. That model is then tested on the test set.

In general, a model is trained on the records in the train set and tested on the records in the test set, each representing a *hold out* record. Using a set of hold out records, represented by the test set in this study, is a common practice [45, 75, 76]. For each run, COSEEKMO uses a different seed for random number generation, resulting in different train and test sets. Hence, each method is trained and tested on different sets. In these experiments, test sets have a constant size, set to 10. This size is chosen as a result of prior research [46] using COSEEKMO. This process is repeated 20 times as described in Section 3.5.1.

One of the other possible options is to use all the data set to train a model and test on the same data set. However, this option is not suitable since in the real world examples, the goal is to assess how a model predicts the effort for a new project [46] not seen before. Also, as Ferens and Christensen [19] describe in their study, the failure to use a hold out sample, like the test set in this work, can overstate a model's accuracy on new examples.

Another option is the k-fold cross-validation, where a set is partitioned into k subsets. Each time, exactly one different subset is used in testing the model while the other k-1 subsets are used in training the model. This option allows for the maximum amount of data for training a model. On the other hand, it is computationally expensive for large data sets. Although only some of the subsets of $COC81$ and $NASA93$ are large, this option is not feasible in COSEEKMO in general. COSEEKMO contains several methods resulting in numerous combined methods. There are also multiple runs in each experiment. Therefore, k-fold cross-validation and other computationally expensive methods such as bootstrapping are not practically possible in this study.

Bouckaert [8] claims that experiments using cross-validation suffer from low replicability. However, this is not the case with COSEEKMO since the random number generation can be replicated exactly when the same seed is used, providing the same results.

3.1.3 Evaluation of Results

The methods in COSEEKMO were evaluated using the non-parametric Mann-Whitney U test, described in detail in Section 3.4.2. The parametric alternatives were not suitable since the error distributions were not normal. The Mann-Whitney U test compares all the methods and combinations used in COSEEKMO and returns a table of win-loss-tie values for each data set or its subsets. Depending on the experiment, one or more of the values reported in such a table were used to decide the best method or methods. In this study, minimizing the number of losses is used for such decisions. However, when two methods have the same number of losses, the one with more wins is ranked higher. Although non-parametric tests are preferred in this study, the parametric alternatives are available as well. COSEEKMO originally used parametric tests only and depended on rejection rules [46] that chose one method over others based on parametric criteria. This is further discussed in Section 3.3.

3.2 Row Pruning

Row pruning is used as a pre-processor in this study. This work explores two distinct row pruning methods with different variations. These are automatic in the way they select similar instances.

This study was initiated partly to explore row pruning methods in order to reduce the presence of outliers as described in Section 2.4. However, after row pruning methods were introduced into COSEEKMO initially, the comparison of methods did not report row pruning as an effective method. Column pruning methods usually outperformed row pruning methods [25]. Nonetheless, soon after the prior study was published, further research showed a systematic error in evaluation criteria used in COSEEKMO and other effort estimation studies in general. This is further discussed in Sections 2.6 and 3.3. The evaluation criteria was the reason for the outliers problem as well, which was the driving force for developing an effective row pruning method. Since outliers were artifacts of the previous evaluation criteria and did not exist in the current results, the first conclusion was that row pruning was not required anymore. Nonetheless, once this evaluation error was resolved, new experiments proved row pruning to be a worthy pre-processor, ranking among the top methods reported by COSEEKMO. This is shown in Section 3.6.

Section 3.2.1 briefly discusses manual stratification for certain few similarities it has with automatic row pruning methods in the remainder of this section. Nearest neighbor, described in Section 3.2.2, uses all the attributes to select similar instances. This is done using a Euclidean distance measure. Locomo is similar to the nearest neighbor method in using all the attributes to select similar instances. However, it uses the local calibration method (described in Section 3.5.5) internally to evaluate the selected instances. Hence, it can be considered a combination method, combining a row pruner and a learner. Therefore, in the remainder of this work whenever Locomo is mentioned in the results as the row pruning method, it is always followed by the local calibration method as its learner, which is used internally. Locomo and its variations are discussed in detail in Section 3.2.3.

3.2.1 Manual Stratification

Boehm et al. advises that effort estimates can be improved via manual stratification (using domain knowledge to select relevant past data) [7]. As described in Section 3.1.1, both $COC81$ and $NASA93$ data sets are composed of different subsets. Each subset can be distinguished from other instances in the main data set using the information stored in the instances of the main data set such as where and when it was gathered. This is a stratification method that makes use of the similarities between instances. It is considered a manual method since the only way to stratify the data is to use the manually gathered information about the data such as the location and time of the project or instance. As a result, each subset contains similar instances. Each subset is then partitioned into train and test sets. This assures that the training and testing are done on data that contains similar instances.

This method is a pre-processor and can be compared to its counterpart, which is described in Section 3.5.1. In the other method, a model is trained using the information in the superset and not in the subset. It is then tested on a test set derived from the subset. Hence, this study can present the results of comparison of these two methods applied to the data as pre-processors. These two methods are applied to every other method, whether a pre-processor or not. These two methods are discussed and their results are analyzed as *train type*s since they are more similar to each other in their approach and less similar to automatic row pruning methods discussed here.

3.2.2 Nearest Neighbor

The ANGEL project [69] inspired this study to explore estimation by analogy as a way to find similarities among train instances, which can be used to find effort estimates for test instances. Nearest neighbor uses the normalized Euclidean distance as a measure of similarity. For two instances X and Y each having m attributes, the normalized Euclidean distance, called distance hereafter, is:

$$\sqrt{\sum_{a=1}^{m}\left(\frac{X_a - Y_a}{Max_a - Min_a}\right)^2}$$

Nearest neighbor takes a train file with n_{train} instances as input and finds the distance between every pair of train instances. It then tries to find the best neighborhood for each train instance i. To do so, it first sorts the remaining instances in ascending order using their distance from i. Then, for each neighborhood k, where $1 \leq k < n_{train}$, it averages the actual efforts of the first k sorted instances. This average value is used as the estimate for the effort of instance i when a neighborhood of size k is used. Finally, depending on the variation of nearest neighbor used, the MRE or AR (also called ME and described in Section 2.6) is calculated for instance i and is added to the sum of MRE's or AR's for neighborhood k. The neighborhood with the minimum sum of MRE or AR is chosen as the best neighborhood.

Once nearest neighbor found the best neighborhood b, it uses this neighborhood to find the estimate for each test instance. Therefore, similar to the training part, the distance between every test instance and all train instances is calculated. For each test instance j, the train instances are sorted in ascending order using their distance from j. Finally, the average of the actual efforts for the first b train instances is used as the estimate of effort for j.

3.2.3 Locomo

Locomo is a row pruner that combines a nearest neighbor method with the local calibration method described in Section 3.5.5. When Locomo was first developed in this work, it only accepted neighborhoods that were set before and would not look for the most suitable neighborhood. Hence, it was given the name static Locomo.

On the other hand, dynamic Locomo tries to find the best neighborhood while processing the train set in order. This neighborhood can change from one train set to another and hence it is dynamic. Although the two methods and their variations are different, they both use the local calibration method in finding effort estimates. The local calibration method uses the k nearest train instances (whether k is a constant or it was found during training) to find that estimate.

Static Locomo

Similar to the nearest neighbor method described in Section 3.2.2, static Locomo uses the normalized Euclidean distance to measure the distance of each test instance j from all train instances. Then, using the provided neighborhood, k, it finds the k closest train instances to j. These train instances are then passed to the local calibration method, which finds an estimate for the test instance j.

This process is repeated until an effort estimate is found for every test instance in the test set. However, since it is not possible to predict the best neighborhood for this method, this study tries a set of neighborhoods $\{5, 10, 20, 40, 80\}$ as the possible best neighborhoods. Therefore, for each pair of train and test sets, each one of these neighborhoods is tried separately and all the results are stored under this method's results when comparing this method against all other methods.

While static Locomo was being developed, the goal was to find a minimum set of neighborhoods that represent other neighborhoods in their performance. Several neighborhoods were tested and in general the neighborhoods in the set $\{5, 10, 20, 40, 80\}$ were shown to provide same or better estimates (depending on the data sets) comparing to all other neighborhoods tested. In addition to performance considerations, these neighborhoods take into account the data set bounds. The largest data set, *NASA*93, has 93 instances. Since in this study a test set always contains 10 instances, a train set can contain at most 83 instances. Therefore, a neighborhood of 80 provides an upper bound for the neighborhoods that can be used in static Locomo. It should be noted that a neighborhood of 83 will use all the train set instances, and hence static Locomo simplifies to the local calibration method. Other neighborhoods were included in this list for similar reasons of representing proper upper and lower bounds for data sets. Overall, these neighborhoods provided an adequate coverage of the data sets used in this study.

Dynamic Locomo

Although the neighborhoods used in static Locomo were generally the best choices, there were several unavoidable, redundant procedures used in static Locomo. Also, the choice of neighborhood was not based on a training process using the train set. Therefore, static Locomo's set of neighborhoods was only suitable for the data sets used in this study and needed revision once new data sets would be used. Therefore, dynamic Locomo was developed.

Dynamic Locomo can process any COCOMO 81 data set and can be easily modified to accept other data sets based on other models used in effort estimation. There are no static neighborhoods used in dynamic Locomo. Therefore, it can find the best neighborhood most suitable for a given train set. Dynamic Locomo uses a similar algorithm to the nearest neighbor algorithm described in Section 3.2.2 when looking for the best neighborhood in the train set.

When given a train set of size n_{train}, it first calculates the distance of every train instance i from all other train instances using a normalized Euclidean distance. Like the algorithm for the nearest neighbor method, dynamic Locomo tries to find the best neighborhood for each train instance i. To do so, it sorts the remaining train instances in ascending order using their distance from i.

Unlike the nearest neighbor method where an average of actual efforts was used, dynamic Locomo uses the local calibration method to find an effort estimate. So, for each neighborhood k, where $1 \leq k < n_{train}$, the first k sorted train instances are passed to the local calibration method along with instance i to find an effort estimate for i. This estimated effort is used (along with the actual effort) to calculate the MRE value for instance i when neighborhood k is used.

At the end of this process, there will be n_{train} MRE values for each neighborhood. The neighborhood that has the lowest median or mean MRE (depending on the variation of dynamic Locomo) is chosen as the best neighborhood b. This neighborhood is used to select the closest b train instances to each test instance j. These train instances along with j are passed to the local calibration method to find an effort estimate for j.

Dynamic Locomo's source code is available at `http://unbox.org/wisp/tags/coseekmo/2.0/locomo/dynamicLocomoMedian.awk`.

3.3 Parametric Tests

Initially, this study used parametric tests to compare different effort estimation methods. As described in Section 2.6.1, parametric tests assume that there is an underlying parameterized distribution in the data. For most parametric tests (including all the ones used in this study before), this underlying distribution is the normal distribution. The parametric tests in this study were used to compare different error distributions such as AR and MRE. As it is shown in Figure 3.1, these error distributions do not represent a normal distribution.

Although it is clear that parametric tests are not suitable for this study in particular, it is beneficial to study such parametric tests in general since they can be used when the underlying distribution is shown to be normal. Therefore, these tests are discussed in detail in the following sections.

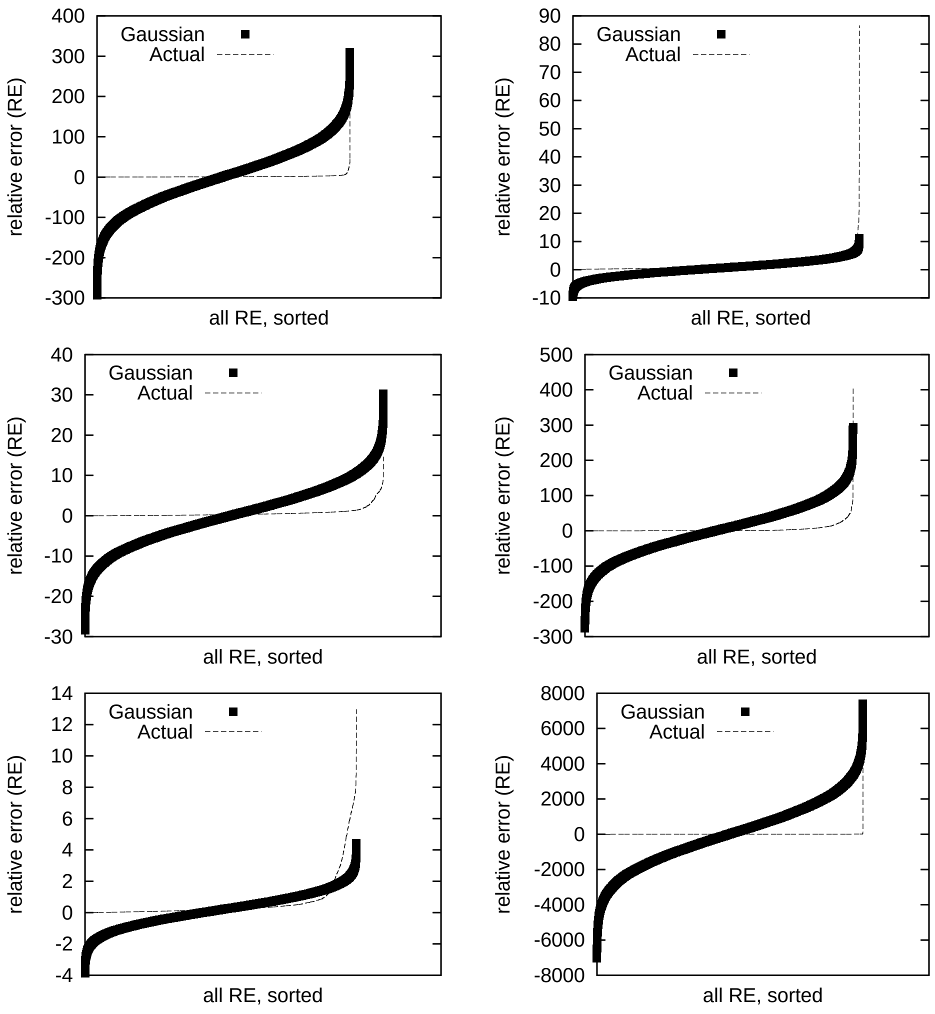

Figure 3.1: Relative error (RE) values seen in experiments with COSEEKMO on six different subsets. Thin lines show the actual values. Thick lines show a Gaussian (normal) distribution that uses mean and standard deviation of the actual values.

3.3.1 General Tests

There are several evaluation criteria discussed in Section 2.6 such as AR and MRE. Previously, this study made use of these criteria in different ways. Mean and standard deviation of these criteria were used extensively. As discussed before, using mean and standard deviation only make sense if the underlying distribution is normal.

3.3.2 Rejection Rules

Previously, effort estimation methods used in this study were compared using rejection rules [46]. These rules and their order are shown in Figure 3.2. There are five rules used in the given order to compare two methods *x* and *y*. The goal is to find the method that performs worse than the other, for every pair of methods in the experiment. The methods that survive this comparison are reported as the best methods. These methods cannot be rejected when comparing them to other methods.

Rule 1 is only applied if the MRE values of *x* and *y* are statistically different. This is tested using a two-tailed t-test at the 95% confidence interval. Therefore, two methods are statistically different if the following inequality holds:

$$\frac{|meanMRE(x) - meanMRE(y)|}{\sqrt{\frac{sdMRE(x)^2}{n(x)-1} + \frac{sdMRE(y)^2}{n(y)-1}}} > 1.96$$

Rule 1 returns the method with the higher mean MRE value as the worse method. However, if the two methods do not have statistically different MRE distributions, other rules may apply.

Rule 2 compares the standard deviation of the MRE values for *x* and *y* and returns the method with higher standard deviation as the worse method. Rule 3 compares the correlation between the predicted and actual values and reports the method with the lower correlation as the worse method. Rule 4 uses the PRED(30) measure on MRE values and reports the method with the lower PRED(30) value as the worse method. Rule 5 compares the number of attributes of the subsets used in both methods and reports the one with more attributes as the worse method (since both

```
FUNCTION worse(x,y)
  IF statisticallyDifferent(x,y) THEN
    IF meanMRE(x) < meanMRE(y)      THEN RETURN y FI  # rule1
    IF meanMRE(y) < meanMRE(x)      THEN RETURN x FI  # rule1
  ELSE
    IF sdMRE(x) < sdMRE(y)          THEN RETURN y FI  # rule2
    IF sdMRE(y) < sdMRE(x)          THEN RETURN x FI  # rule2

    IF correlation(x) < correlation(y)  THEN RETURN x FI  # rule3
    IF correlation(y) < correlation(x)  THEN RETURN y FI  # rule3

    IF pred(x) < pred(y)            THEN RETURN x FI  # rule4
    IF pred(y) < pred(x)            THEN RETURN y FI  # rule4

    IF $|Subset(x)| < |Subset(y)|$  THEN RETURN y FI  # rule5
    IF $|Subset(y)| < |Subset(x)|$  THEN RETURN x FI  # rule5
  FI
RETURN 0  # if no reason to return true
```

Figure 3.2: COSEEKMO's rejection rules.

have similar performances).

These rules were obtained from an earlier study using COSEEKMO by Menzies et al. [46]. They mention that their rejection rules can reproduce Boehm's 1981 COCOMO 81 analysis. However, this study achieved the same results when the order of the rules were modified [25]. Rule 2 was also modified from their study to use the standard deviation only, which did not change the results. According to their study, lower priority rules were used with a much lower frequency when methods were compared pairwise.

3.3.3 Instability

Although the parametric tests used in this study were able to reproduce the results in the literature, they lacked the stability when reporting the best methods. Different methods performed differently for different data sets and their subsets. This is clearly shown in the previous work [25] prior to this study, reproduced here in Figures 3.3 and 3.4.

Each figure shows three parametric tests in three separate graphs. A vertical line in any set of graphs in these figures shows the results of different methods (shown in the legend) on the same subset of a data set. As it can be seen from these results, different methods perform differently across different data sets and from one parametric test to another. Hence, it is hard to compare different methods across different data sets. This would be even harder when the results were compared from one run of the experiment to another, where two runs only differed in their random number generator's seeds. The source of this problem could be considered the parametric tests. These tests were not suitable for this study since the error distributions were not normal. However, this problem was solved by using non-parametric tests, discussed in Section 3.4.

3.4 Non-Parametric Tests

This study originally used parametric tests. However, as explained before, these tests make assumptions about the underlying distributions. Such assumptions cannot be guaranteed in every experiment and as Figure 3.1 shows, the experiments done in this study do not have a normal distributions required for the parametric tests done before. Therefore, non-parametric tests replaced parametric tests in this study.

Non-parametric tests do not assume an underlying distribution and therefore can be used in this study. Among different non-parametric tests, the Mann-Whitney U test, also known as the Wilcoxon rank-sum test, is the most suitable test for comparing the methods used in this study. This test is discussed in Section 3.4.2. However, before this test was chosen, other non-parametric tests were explored and studied in detail. One such test is the median test, discussed in Section 3.4.1.

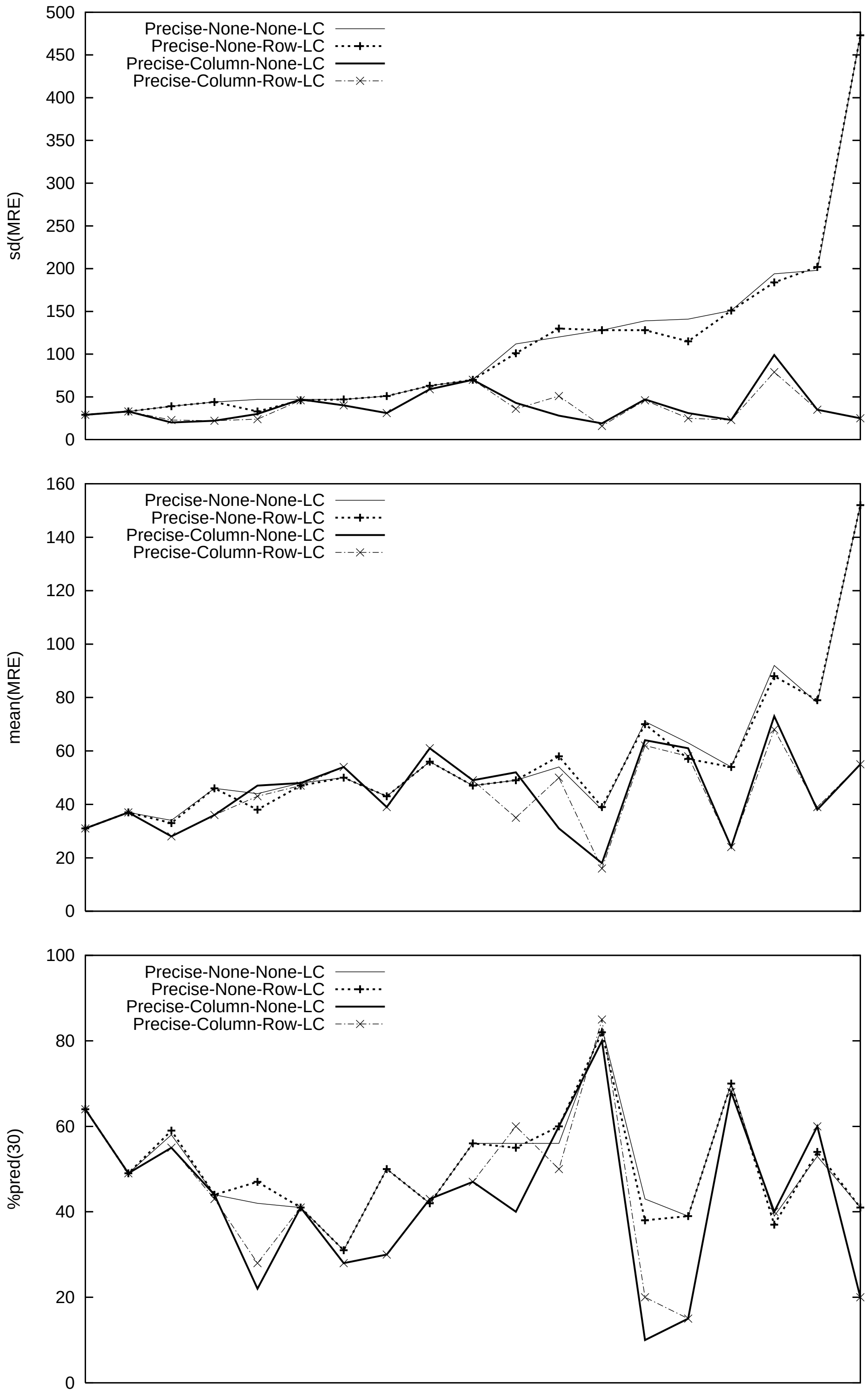

Figure 3.3: Effects of different pruning methods showing instability in using parametric tests.

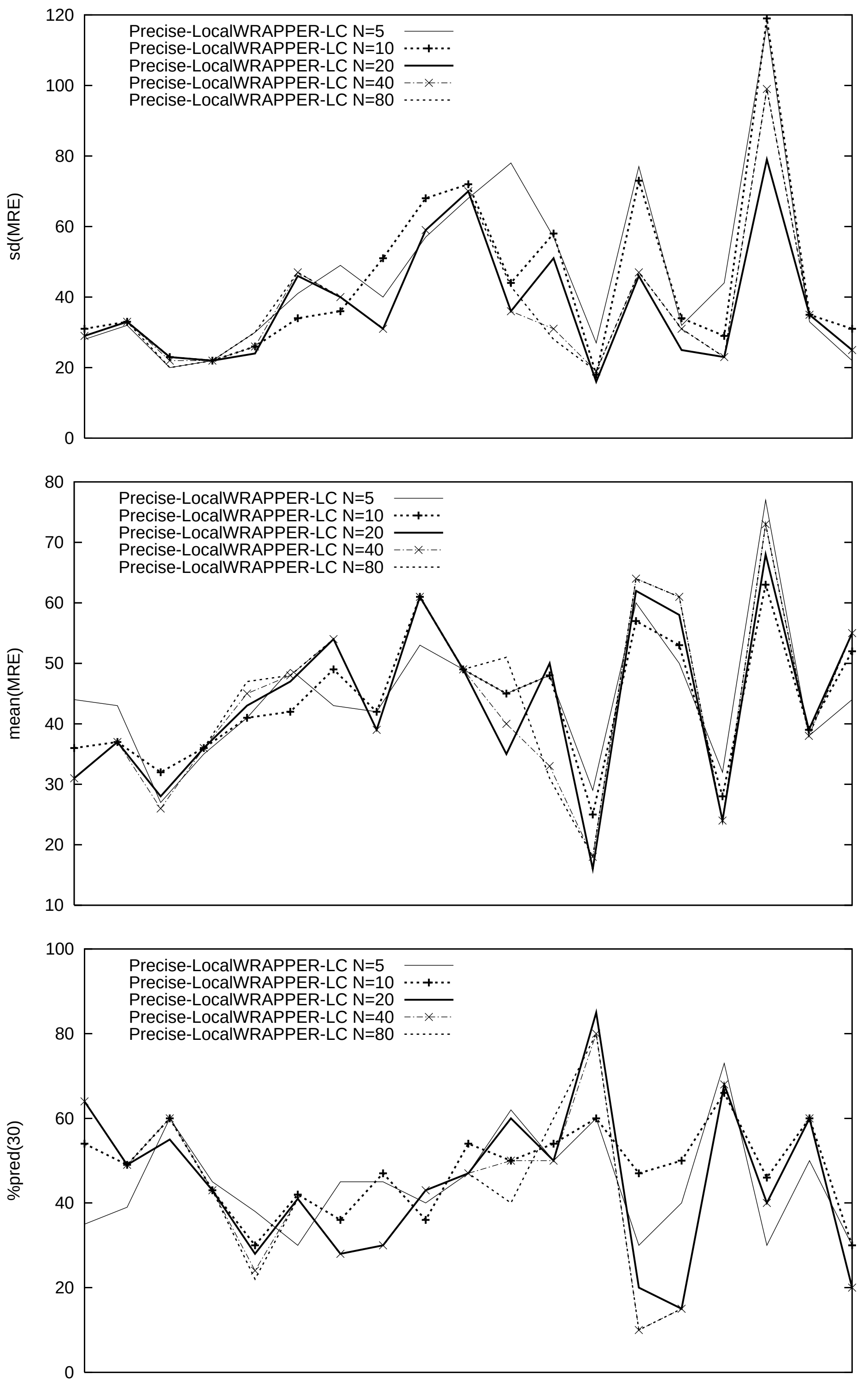

Figure 3.4: Effects of different neighborhoods showing instability in using parametric tests.

3.4.1 Median Test

The median test used in this study compared two different distributions using their median and spread. This test sorts the error estimates, using AR, MRE, etc. It then reports value in the middle as the median and the difference between the values at one-third and two-third of the sorted list as the spread. Finally, the method with the lowest median and spread was reported as the best method.

Although this test was simple, it did not produce stable and conclusive results. Most of the methods reported as the winners in one experiment were not reported as the winners in another experiment with a different random number generator seed. Therefore, a stronger test was needed, which led this research to the Mann-Whitney U test.

3.4.2 Mann-Whitney U Test

This test was developed independently by Wilcoxon in 1945 and Mann and Whitney in 1947 [71, p109-118]. Therefore, it is known by two different names: the Wilcoxon rank-sum test and the Mann-Whitney U test. Wilcoxon proposed this test as well as the Wilcoxon signed-rank test in the same paper [74]. The Wilcoxon signed-rank test is suitable for cases where the two samples are related or are two replications of the same experiment on the same sample. Since this study explores different samples with different methods, the Wilcoxon signed-rank test could not be used.

On the other hand, the Wilcoxon rank-sum test requires the two samples to be statistically independent. This is the case in this study, where two methods applied to a data set create two statistically independent error samples. However, the Wilcoxon rank-sum test only explores equal sample sizes [42]. This could not be assured in this work since certain pre-processor methods could create error samples with different sizes depending on their heuristics. Nonetheless, the test proposed by Mann and Whitney, which is an extension of the Wilcoxon rank-sum test, resolved this problem. As a result, this study uses the Mann-Whitney U test, or the MWU test hereafter.

The MWU test assesses the null hypothesis, which states that the two samples come from identical distributions. Also, the MWU test requires the samples to have continuous measurements [42]. For two samples X and Y, the MWU test first ranks the two samples. To do so, the samples are sorted into one large sample while maintaining the information about which sample each value came from. Each value is then ranked with a number between 1 and $|X|+|Y|$. If there are two or more identical values, the average of the ranks of these values is assigned as their identical ranks. For example, if k identical values $v_1,...,v_k$ have ranks $r_1,...,r_k$, they all get the same rank of:

$$\frac{\sum_{i=1}^{k} r_i}{k}$$

It is clear that ranking the values removes the effects of the outliers present in certain tests such as the parametric tests. A value large enough to be considered an outlier will have a larger rank than the previous non-outlier value in the sorted list. However, this rank is only larger by a constant $c < k$. Once all the values of both samples are ranked, the MWU test calculates the sum of the ranks for all the values in each sample. Therefore, for a sample X, the sum of the ranks is:

$$R_X = \sum_{r_i \in X} r_i$$

R_Y is calculated the same way. Using the sum of the ranks for each sample, the U statistic is calculated as follows, where $n_X = |X|$:

$$U_X = R_X - \frac{n_X(n_X+1)}{2}$$

U_Y is calculated the same way. In addition to U_X and U_Y, the mean and standard deviation of U must also be found using the following equations, where m_U is the mean of U, σ_U is the standard deviation of U, and $n_Y = |Y|$:

$$m_U = \frac{n_X n_Y}{2}$$

$$\sigma_U = \sqrt{\frac{n_X n_Y (n_X + n_Y + 1)}{12}}$$

Finally, the following normal approximation can be used to calculate the standard normal deviate, z, also known as the standard score:

$$z = \frac{U - m_U}{\sigma_U}$$

Since there are two samples X and Y, there will be two standard scores, z_X and z_Y. These values are always equal in magnitude and opposite in sign. Therefore, they are simply identified by z, where $z = |z_X| = |z_Y|$. In order to assess the null hypothesis, a confidence interval is used. This study uses two different confidence intervals. At 95% confidence interval, $z <= 1.96$ must hold to accept the null hypothesis. At 99% confidence interval, $z <= 2.576$ must hold to accept the null hypothesis.

Given two methods m_X and m_Y, the acceptance or rejection of the null hypothesis can only identify these methods' performance (based on their error samples X and Y) as being identical or not respectively. In the case of non-identical methods, the MWU test does not help in identifying which method performed better. Nonetheless, in comparison of different methods it is more interesting to find the ones that performed best. Therefore, in case of two non-identical samples, which happens when the null hypothesis is rejected, another non-parametric test can augment the MWU test. The simplest and most useful non-parametric test is to compare the median ranks of the two samples after the MWU test considered them non-identical. If the performance measure used in the MWU test is an error estimate (such as AR, MRE, etc.), a lower value is better. Therefore, if sample X has a lower median rank than sample Y, then method m_X is considered better than method m_Y. On the other hand, if the performance measure used in the MWU test is considered better when it has a higher value, then method m_Y is considered better than method m_X. Finally, if both samples have the same median rank, their corresponding methods are considered identical.

To summarize, the MWU test augmented with the median rank test can score two methods as follows (when a lower value in a sample is considered better):

- If the null hypothesis holds, both methods get a *tie*.
- If the null hypothesis is rejected and the median rank of X is lower than the median rank of Y, method m_X gets a *win* and method m_Y gets a *loss*.
- If the null hypothesis is rejected and the median rank of X is higher than the median rank of Y, method m_X gets a *loss* and method m_Y gets a *win*.
- If the null hypothesis is rejected and the median rank of X is equal to the median rank of Y, both methods get a *tie*.

It is clear that if a higher value in a sample is considered better, the above assignments of wins and losses must be switched.

The slowest part of the described algorithm for the MWU test augmented with the median rank test is the sorting done in the first step. There are two samples of size n_X and n_Y and the combined sample of size $n_X + n_y$ must be sorted. The rest of the algorithm has a time complexity of $\theta(n_X + n_Y)$. Therefore, depending on the sorting algorithm, the MWU test's algorithm can be implemented efficiently. An example of the MWU test is provided in Figure 3.5.

It is clear that the MWU test can only compare a pair of methods. In this study, there are several methods used in a single experiment. Therefore, the MWU test was applied to each distinct pair of methods once. The resulting number of wins, ties, and losses for each method were used to identify the best method or methods in each experiment. Since N methods are compared in a pairwise fashion, a method is compare against $N-1$ other methods. Hence, the sum of the number of wins, ties, and losses for each method is equal to $N-1$. In order to identify the best method

There are two samples A and B with $|A| = 5$ and $|B| = 6$:

$$A = \{5,7,2,0,4\}$$
$$B = \{4,8,2,3,6,7\}$$

They are sorted as follows:

Samples	A	A	B	B	A	B	A	B	A	B	B
Values	0	2	2	3	4	4	5	6	7	7	8

The rankings are:

Samples	A	A	B	B	A	B	A	B	A	B	B
Values	0	2	2	3	4	4	5	6	7	7	8
Ranks	1	2.5	2.5	4	5.5	5.5	7	8	9.5	9.5	11

The sum and median of A's ranks are:

$$R_A = 1+2.5+5.5+7+9.5 = 25.5$$
$$median_A = 5.5$$

and the sum and median of B's ranks are:

$$R_B = 2.5+4+5.5+8+9.5+11 = 40.5$$
$$median_B = (5.5+8)/2 = 6.75$$

The U statistic for A and B is:

$$U_A = 25.5 - 5*6/2 = 10.5$$
$$U_B = 40.5 - 6*7/2 = 19.5$$

Mean and standard deviation of U are:

$$m_U = 5*6/2 = 15$$
$$\sigma_U = \sqrt{5*6*(5+6+1)/12} = 5.477$$

Finally, the z values for A and B are:

$$z_A = (10.5-15)/5.477 = -0.82$$
$$z_B = (19.5-15)/5.477 = 0.82$$

Since $|z_A| = |z_B| = 0.82 < 1.96$, the null hypothesis holds and thus samples A and B are considered identical (at the 95% significance level). In this case, both methods corresponding to A and B get a tie. If the null hypothesis did not hold, since $median_A < median_B$, A's method gets a win and B's method gets a loss (assuming that a low value in the sample is considered better).

Figure 3.5: An example of the Mann-Whitney U test.

or methods, this work explored using wins, ties, losses, and different combinations of these three numbers such as the difference between the number of wins and losses.

Since the goal is to find the best method or methods, the remaining methods can be discarded. Therefore, an insightful metric is the number of losses. If this is non-zero, then there is a case for discarding that method. However, in case of several methods having the same number of losses, the number of wins plays a role in making a decision. As a results, if two or more methods have the same number of losses, the ones with more wins are considered better. Using only the number of losses (and wins if necessary) might render the calculation of the number of ties redundant. However, before the number of wins and losses can be found, a decision must be made about the acceptance or rejection of the null hypothesis, which automatically results in calculating the number of ties.

In general, the MWU test described here can be used to compare any number of methods using any performance measure, such as error estimates. It should be noted that, wins, ties, losses, or any combination of these numbers cannot necessarily be the suitable identifier of superiority of a method over others in every study. As an example, if in a study the number of wins was to be used as the way of identifying the best method, it might not be a suitable measure for other studies. This is because if a method ties with the majority of other methods and does not lose to any, it can be considered a superior method since it performs at least as well as any other method. Therefore, each study requires an in-depth understanding of its performance measure and possible ways of interpreting the MWU test's results.

As another example, the confidence interval should be used with great care. Since the critical value for the confidence interval of 99% is greater than that of 95% confidence interval, it is more likely that the null hypothesis holds. Therefore, there are more ties possible and as a result, the number of wins and losses will be reduced. Hence, if the number of losses or wins were to be used to identify the best methods, such identification could become harder to achieve, resulting in a possible confusion in the comparison. Thus, it is helpful to understand these details about the

MWU test before applying it. The complete source code of the MWU test along with instructions are available at `http://unbox.org/wisp/tags/mwu/1.0/lib/mwu.awk`.

3.5 COSEEKMO

COSEEKMO is a tool used in effort estimation experiments. Menzies et al. designed it initially and discussed its several uses [25, 39, 43, 44, 46]. COSEEKMO's goal is to find the best group of methods for effort estimation on COCOMO-style data sets. Currently, COSEEKMO uses COCOMO 81 data sets for experimentation. However, it can easily be extended to use COCOMO II data sets. Shepperd considers COSEEKMO's main limiting factor to be the required conformity of the collected data to the COCOMO model, although he considers the motivation for COSEEKMO "attractive" [64]. At the time of developing COSEEKMO (and until now) COCOMO data sets were the only accessible and publicly available data sets for effort estimation experiments. However, it is possible (and rather simple) to modify COSEEKMO to work with other models as well.

There are several different methods currently present in COSEEKMO. Some of these methods are simple while others use complex algorithms for effort estimation. Usually, more complex methods are preferred since they are believed to explore more possibilities and provide the best effort estimate in this case. However, as Menzies et al. suggest, "it is good practice to benchmark elaborate or resource intensive techniques against simpler alternatives [45]." According to Cohen's argument [14], it is possible that simpler methods can achieve similar results to sophisticated methods and hence the latter is superfluous in such cases. As an example, they mention Holte's famous study where he compared classification rules and showed how simple ones usually performed well [22]. Similarly, as presented in Section 3.6, in every case, the more complex methods are superfluous and simpler ones perform at least as well as any other method.

Since its introduction, COSEEKMO has been modified and improved constantly to explore a wider range of effort estimation methods. For example, it is possible that the combination of any number of effort estimation methods can provide better effort estimates. Therefore, COSEEKMO explores all such combinations. Also, the evaluation methods of COSEEKMO have been modified to include non-parametric tests such as the MWU test (introduced in Section 3.4.2).

Although the methods explored in COSEEKMO are integrated for increased performance, they can be used individually as well. For example, NASA's Jet Propulsion Laboratory uses the dynamic Locomo method (discussed in Section 3.2.3), which is currently a method integrated into COSEEKMO [2]. This satisfies one of the main goals of COSEEKMO, which is to find a group of best methods to be used in effort estimation practices. Also, new effort estimation and evaluation methods can be easily added to COSEEKMO for analysis and comparison. Cocomin [2] is an effort estimation method that was added to COSEEKMO in this work. The MWU test is an example of evaluation methods added to COSEEKMO. Such additions increased the stability of results in COSEEKMO, which can be seen in Section 3.6.

This section provides an overview of COSEEKMO's algorithm (Section 3.5.1), numeric estimation methods (Section 3.5.2), pre-processors (Sections 3.5.4 and 3.5.3), learners (Section 3.5.5), and combinations of methods and experiments (Section 3.5.6).

3.5.1 Algorithm

The algorithm for COSEEKMO is presented in Figure 3.6. In this algorithm, *Data* represents all 19 subsets of *COC*81 and *NASA*93 data sets. *Runs* is a constant representing the number of runs for each data set. Previous studies used 30 runs [46]. However, in this work, 20 runs showed the same stability. Therefore, COSEEKMO uses 20 runs in this study.

As discussed in Section 3.1.2, in each run and for each subset, 10 instance are chosen at random to create a test set. Then, based on the *train type*, a train set is generated. One train set is generated using all the instances in the subset except for the ones in the test set. This method is called the *subset* train type. The other train set is generated using all the instances in the superset except for the ones in the test set. This method is called the *superset* train type. These train sets are used

```
FOR datum IN Data DO
   FOR run IN Runs DO
      FOR train type IN {superset,subset} DO
         partition datum into train and test sets          # train set is created using train type
         FOR numeric estimation IN {precise,proximal} DO
            apply numeric estimation to train and test sets
            apply methods and their combinations to train and test sets
         DONE
      DONE
   DONE
DONE
run evaluation methods on the results
```

Figure 3.6: COSEEKMO's algorithm.

separately to train different models later.

Once the numeric estimation methods (discussed in Section 3.5.2) are applied to the train and test sets, different methods or combinations of methods are applied to each train set and the trained models are used to generate effort estimates. These methods are categorized into two groups of pre-processors (column pruners and row pruners) and various learners, all discussed in the following sections. Finally, an evaluation method compares the effort estimates generated from each method or combination of methods and reports the best methods. This study used the MWU test as described in Section 3.4.2.

3.5.2 Numeric Estimation Methods

COSEEKMO uses COCOMO data sets for its effort estimation experiments. As described in Section 2.2.1, there are several cost drivers in the COCOMO 81 model, represented in ordinal values in a rating scale. Table 2.2 provided the names of these cost drivers and Table 2.3 provided the numeric values of the effort multipliers matching their ordinal values in the rating scale. In this study, the numeric estimation method that uses these effort multipliers is called the *precise* numeric estimation method.

	Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Upper (increase these to decrease effort)	ACAP	1.2	1.1	1.00	0.9	0.8	
	PCAP	1.2	1.1	1.00	0.9	0.8	
	AEXP	1.2	1.1	1.00	0.9	0.8	
	MODP	1.2	1.1	1.00	0.9	0.8	
	TOOL	1.2	1.1	1.00	0.9	0.8	
	VEXP	1.2	1.1	1.00	0.9		
	LEXP	1.2	1.1	1.00	0.9		
Middle	SCED	1.2	1.1	1.00	1.1	1.2	
Lower (increase these to increase effort)	DATA		0.9	1.00	1.1	1.2	
	TURN		0.9	1.00	1.1	1.2	
	VIRT		0.9	1.00	1.1	1.2	
	STOR			1.00	1.1	1.2	1.3
	TIME			1.00	1.1	1.2	1.3
	RELY	0.8	0.9	1.00	1.1	1.2	
	CPLX	0.8	0.9	1.00	1.1	1.2	1.3

Table 3.4: The proximal COCOMO 81 effort multiplier values.

These values were generated by Boehm, using his project experience [4]. It is hard to regenerate these values for each new organization requiring effort estimation since there is usually not enough relevant data available within most organization. Therefore, another approach is to use a *proximal* numeric estimation of the values in Table 2.3, as suggested by Menzies et al. [46]. These proximal values are generated by mapping the precise values into two approximate straight lines. One line has a negative slope and is used for ACAP, PCAP, AEXP, MODP, TOOL, VEXP, and LEXP cost drivers. The other one has a positive slope and is used for DATA, TURN, VIRT, STOR, TIME, RELY, and CPLX cost drivers. A third proximal set of values is used for the SCED cost driver, which has identical negative and positive slopes. The proximal effort multiplier values are shown in Table 3.4. Both precise and proximal values are used in COSEEKMO.

3.5.3 Column Pruning Pre-Processors

COSEEKMO uses several column pruning methods as pre-processors. Column pruning is useful in removing columns (also known as features, attributes, or variables) of a data set that create noise, mostly in terms of deviations, in the model. In order to study the effects of column pruning, COSEEKMO compares several column pruners (including no column pruner) as pre-processors. These column pruners differ mostly in the thoroughness of their search algorithms.

Wrapper

Wrapper [37] is a standard best-first search through the space of possible features. At worst, the Wrapper must search the whole space, or all the subsets of features F in a data set, which is an exponential number 2^F. However, a simple best-first heuristic makes Wrapper practical for effort estimation. At each step of the search, all the current subsets are scored by passing them to a target learner. Two different target learners, LSR and M5p, are used in this study. These learners are further discussed in Section 3.5.5. Wrappers used in COSEEKMO use the forward search, meaning that they start with an empty subset of features and add useful features. If an added feature does not help the subset containing that feature score better than a smaller or different subset of the same size, that feature gets a negative point. Once a feature has a certain number of negative points, it is removed from the subset of features reported by the Wrapper at the end of its algorithm. The Wrappers used in COSEEKMO remove a feature if it has more than 5 negative points. The remaining features are reported by the Wrapper and are used to train and test a model later by COSEEKMO. When LSR or M5p are used as target learners, the data sets are usually logged and the results are unlogged. Therefore, the Wrappers used in COSEEKMO with LSR and M5p as target learners also log the data. However, to study the effects of logging, non-logged data sets are also used and compared with logged data sets.

Local Wrapper

Local Wrapper, developed by Chen et al. [12, 46], also uses a thorough search. However, it uses the local calibration method as its target learner. This further specializes Local Wrapper for effort estimation.

Cocomin

Cocomin, developed by Baker [2], is another column pruning method that uses a less thorough search algorithm. Cocomin is a near-linear-time column pruner. Since it uses some heuristic criteria to select features, it does not explore all the subsets and hence, Cocomin is a very fast column pruner that "runs in $O(F.log(F))$ for the sort and $O(F)$ for the exploration of the selected features [2]."

Baker describes Cocomin as follows:

> Cocomin is defined by the following operators:
>
> $$\{sorter, order, learner, scorer\}$$
>
> The algorithm runs in linear time over a *sorted* set of features, F.
>
> Cocomin pre-sorts the features on some heuristic criteria. Some of these criteria, such as standard deviation or entropy, are gathered without evaluation of the target learner. Others are gathered by evaluating the performance of the learner using only the feature in question plus any required features, such as *KLOC* for COCOMO, to calibrate the model.
>
> This search can be *order*ed in one of three ways:
>
> - A "backward elimination" process starts with all features F and throws some away, one at a time.
> - A "forward selection" process starts with one feature and adds in the rest, one at a time.
> - "Both" forward and backward searches are run separately and the best performing result is chosen.

> Regardless of the search order, at some point the current set of features $F' \subseteq F$ is passed to a *learner* to generate a performance *score* by applying the model learned on the current features to the train set. Cocomin returns the features associated with the highest score.
>
> After the features are ordered, each feature is considered for backward elimination, or forward selection if chosen, in a single linear pass through the feature space, F. The decision to keep or discard the feature is based on an evaluation measure generated by calibrating and evaluating the model with the training data [2].

There are three variations of Cocomin used in COSEEKMO as a column pruning pre-processor. The first one

- sorted the features by the highest median MRE;
- used a backward elimination search order;
- learned using the local calibration method;
- scored using mean MRE.

The second one

- sorted the features by their native order (simply meaning that it did not do any additional sorting);
- used a backward elimination search order;
- learned using the local calibration method;
- scored using mean MRE.

Finally, the third one (which was an older version of Cocomin)

- sorted the features by their native order (simply meaning that it did not do any additional sorting);
- ran both forward selection and backward elimination search orders to find the best;
- learned using the local calibration method;
- scored using mean MRE.

These variations only differ in the internal settings. These settings were decided through several experiments. Some of the results of these experiments are available in Table 3.7 of [2]. The experiments performed using COSEEKMO over all such settings suggested the same results and hence, the above three settings were selected.

Theoretically, a Wrapper (with target learners such as local calibration, LSR, and M5p) uses an exponential-time search algorithm and is more thorough. Hence, it is believed to be more useful than a simpler method such as Cocomin. However, as shown later in Section 3.6, Cocomin is superior to Wrapper pre-processors.

3.5.4 Row Pruning Pre-Processors

Row pruning methods are another group of pre-processors that are used in COSEEKMO. After the addition of new column pruning methods discussed before, several row pruning methods were added to COSEEKMO in order to remove instances or projects in the data sets that could create noise in the trained models.

There are two types of row pruning used in COSEEKMO. The *manual* method, called manual stratification, uses the information stored in each instance and divides each data set into several subsets. This is discussed in Sections 3.2.1 and 3.5.1 with details of the method and the subset and superset train types. However, a better comparison of pre-processors compares this pre-processor, also called the subset train type, with another pre-processor, called the superset train type, for their similarities.

The *automatic* row pruning methods include the nearest neighbor method, described in Section 3.2.2, and Locomo, described in Section 3.2.3. Both static Locomo and dynamic Locomo are used in COSEEKMO. The nearest neighbor method has two variations. One of the variations uses the MRE in training the model and the other uses AR. The static Locomo method uses different neighborhoods in the set $\{5,10,20,40,80\}$ as described before. Dynamic Locomo has two variations. One variation uses the mean of MRE's in training the model and the other uses the median of MRE's. Overall, Locomo is superior to the nearest neighbor method as shown in Section 3.6.

3.5.5 Learners

Several learners are used in COSEEKMO. Learners are usually applied to the processed data after pre-processors are used. Learners train and test on the data available in order to find the best performing method. In COSEEKMO, some learners must be applied without a previously applied pre-processor. These learners, including Cocomin and Cocomost, are actually pre-processors used as learners for simplicity. COSEEKMO allows for such learners. On the other hand, other learners such as local calibration, LSR, and M5p can be applied with or without pre-processors applied to the data. All the learners used in COSEEKMO are discussed below.

Linear Regression

Linear regression assumes that the data can be approximated by one linear model that includes lines of code (*KLOC*) and other features $f \in F$ seen in a software development project:

$$effort = \beta_0 + \sum_i \beta_i \cdot f_i$$

Linear regression adjusts β_i to minimize the error. Boehm argues that effort is exponential on

KLOC [4]:

$$effort = a \cdot KLOC^b \cdot \prod_i \beta_i$$

where *a* and *b* are domain-specific constants. Such exponential functions can be learned via linear regression after they are converted to the following linear form:

$$log(effort) = log(a) + b \cdot log(KLOC) + \sum_i log(\beta_i)$$

In general, it is useful to transform from a more complex space (in this case, the exponential model) to a simpler one (in this case, the linear model) since it is easier to explain and work with the simpler space. In this case, it is especially useful to find *a* and *b* values when dealing with a linear model rather than solving an exponential equation for *a* and *b*. The majority of the methods in COSEEKMO transform the data in this way. This approach is also suggested by Myrtveit et al. [54] in their study involving linear regression. It is necessary to unlog the estimates returned from each method after the trained model has been tested.

Weka [76] provides the functionality needed for linear regression. This is called LSR and COSEEKMO uses it through Weka.

Model Trees

Model trees are a generalization of linear regression. Instead of fitting the data to one linear model, model trees learn multiple linear models and a decision tree that decides which linear model to use. Model trees are useful when the projects form regions and different models are appropriate for different regions. COSEEKMO includes the M5p model tree learner defined by Quinlan [60]. Weka [76] provides the functionality needed for M5p. As discussed in the case with LSR, M5p also logs the data in order to transform the space and therefore, the estimate must be unlogged.

Local Calibration

Local calibration is a specialized form of linear regression developed by Boehm [4, p526-529]. Local calibration assumes that project effort is exponential on *KLOC*:

$$effort = a \cdot KLOC^b \cdot \prod_i \beta_i$$

Table 2.3 shows the β_i values recommended by Boehm (with the names on the left hand side defined in Table 2.2). When β_i values are used in the above equation, they yield estimates in months where one month is 152 hours (and includes development and management hours). To operate, local calibration linearizes the exponential equation to generate

$$log(effort) = log(a) + b \cdot log(KLOC) + \sum_i log(\beta_i)$$

Linear regression would try to adjust all the β_i values. This is not practical when training on a very small number of projects. Hence, local calibration fixes the β_i values while adjusting a and b values to minimize the prediction error.

COSEEKMO uses local calibration on the training set to find a and b values and applies those to the test set to find estimates. However, COSEEKMO also applies the three values for a and b as suggested by Boehm for embedded, organic, and semidetached software modes. These values can be found in Table 2.1.

Cocomin

Cocomin (described in Section 3.5.3) is a column pruning method with local calibration as its target learner. Although COSEEKMO usually uses Cocomin as a pre-processor, Cocomin can also be used as a learner without any column or row pruning pre-processors applied to it. COSEEKMO uses three variations of Cocomin as explained in Section 3.5.3.

Cocomost

Cocomost is another column pruning pre-processor with local calibration as its target learner. It was developed by Baker [2] and is used as a learner in COSEEKMO without any column or row pruning pre-processors applied to it. Cocomost evaluates all of the possible subsets of features. There are several evaluation methods used, similar to the evaluation methods in Cocomin. As shown in Section 3.6, Cocomost performs similar to Cocomin although Cocomost explores more subsets. A detailed analysis of Cocomost and Cocomin is available in [2].

3.5.6 Combinations of Methods And Experiments

COSEEKMO can use every possible combination of pre-processors and learners as well as numeric estimation methods and train types to create a combination of methods. Certain methods can also be used individually, such as Cocomin and Cocomost learners. Each method, whether combined or single, is evaluated against every other method using the MWU test. The results of such comparisons in the experiments using COSEEKMO are presented in Section 3.6. The total number of methods can be found by calculating the possible combinations. There are 8 column pruning pre-processors applied to 8 row pruning pre-processors and learners. Therefore, there are a total of $8 * 8 = 64$ combinations of column pruning pre-processors, row pruning pre-processors, and learners. All the 8 row pruning pre-processors and learners can be applied without a column pruning pre-processor. In addition, 6 more learners are also applied without a column pruning pre-processor. Therefore, there are $8 + 6 = 14$ more combinations possible that do not use any column pruning pre-processor. Overall, $64 + 14 = 78$ combinations of pre-processors and learners are used in COSEEKMO. Since there are 2 train types and 2 numeric estimation methods applied to each combination, a total of $2 * 2 * 78 = 312$ combinations are used in COSEEKMO.

3.6 Results And Discussions

This section offers the results of the current study in detail. However, this study have been under careful review for a relatively long time since, as shown later, it shows stability in the results that is not reported elsewhere in the literature. Many runs of the previous versions of the experiments were analyzed in detail before and showed the same conclusions as this study's results. The more interesting fact is that, as the experiments evolved and were expanded to include more methods, the conclusions did not change. For example, comparing to the most recent version of the experiments using COSEEKMO, the train types were added, which doubled the number of combinations possible. Although this could have potentially changed the conclusions from the previous versions of the experiment, it had no effect on them. Overall, these results show a great amount of stability as the experiments evolved, and as they are discussed in this section, they will provide a clear sense of how to arrive at the final conclusions.

The results of the experiments using COSEEKMO are extensive. There are 19 subsets of data sets used in this study and 312 combinations of methods possible. Therefore, reporting the results of these methods applied to all these subsets cannot be done using simple win-loss-tie tables obtained from the MWU test. Also, understanding such tables is a very time-consuming task. In addition, these results are analyzed using three different evaluation criteria (AR, MER, and MRE), adding to the amount of analysis required. Hence, it is essential to categorize these findings through analyzing the results across different subsets and data sets, different evaluation criteria, and different runs. This will provide better perspectives about these biases and will offer a gradual path toward selecting the best effort estimation methods in COSEEKMO.

COSEEKMO was used with its 312 methods to generate the results in one representative run. Such a run is called a complete run. The results of a random complete run is provided in Sections 3.6.1, 3.6.2, and 3.6.3. Each section provides a different analysis of the results. In order to simplify the results and focus on the more useful methods, the training type was reduced to subset and half of all the combinations of these methods ($312/2 = 156$) were used. Also, the proximal numeric estimation method was removed from the results and only $156/2 = 78$ methods were reported. Such a run is called a simple run. Simple runs take less time to complete and are relatively easier to analyze. The justification for simpler runs as well as the results of such a run are provided in Section 3.6.4. Finally, a detailed look at the best and worst performers are provided in Section 3.6.5, where an additional 4 runs of the experiment are reported for the 8 representative methods selected from the list of 78 methods.

3.6.1 Top 10 Methods For Each Subset Across Three Evaluation Criteria

This is the most complete set of results since it explores the findings across different subsets and different evaluation criteria. The top 10 methods for each subset is reported individually in Tables 3.5 through 3.61. This contains the top 3% of the methods that have the fewest number of losses (and the most number of wins in case of the same number of losses). The justification for not showing more results is offered in Section 3.6.3. These detailed tables help in identifying the methods suitable for each subset. The reason for reporting these individual tables is that, one method may be more suitable for a subset while it might perform worse for other subsets. However, such a method may be ignored when the results are reported in any other fashion. It should be noted that the total number of losses (as well as wins and ties) possible is 311 for each method.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	NewCocominNative	StaticLocomo	lc	105	206	0
Superset	precise	NewCocominNative	StaticLocomo	lc	105	206	0
Subset	precise	None	StaticLocomo	lc	106	205	0
Superset	precise	None	StaticLocomo	lc	106	205	0
Subset	precise	NewCocominMedian	StaticLocomo	lc	107	204	0
Subset	precise	OldCocomin	StaticLocomo	lc	107	204	0
Superset	precise	OldCocomin	StaticLocomo	lc	107	204	0
Superset	precise	NewCocominMedian	StaticLocomo	lc	108	203	0
Subset	precise	None	DynamicLocomoMedian	lc	114	197	0
Superset	precise	None	DynamicLocomoMedian	lc	114	197	0

Table 3.5: Top 10 methods for subset 1 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	NewCocominMedian	None	lsr	122	189	0
Superset	precise	NewCocominNative	None	lsr	122	189	0
Superset	precise	OldCocomin	None	lsr	122	189	0
Superset	precise	None	None	lsr	123	188	0
Superset	precise	None	StaticLocomo	lc	125	186	0
Superset	proximal	NewCocominMedian	None	lsr	136	175	0
Superset	proximal	NewCocominNative	None	lsr	136	175	0
Superset	proximal	OldCocomin	None	lsr	136	175	0
Superset	precise	NewCocominMedian	StaticLocomo	lc	139	172	0
Superset	precise	NewCocominNative	StaticLocomo	lc	139	172	0

Table 3.6: Top 10 methods for subset 2 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	None	StaticLocomo	lc	42	269	0
Superset	precise	NewCocominMedian	StaticLocomo	lc	46	265	0
Superset	precise	NewCocominNative	StaticLocomo	lc	46	265	0
Superset	precise	OldCocomin	StaticLocomo	lc	46	265	0
Subset	precise	None	None	lc	77	234	0
Superset	precise	None	DynamicLocomoMean	lc	84	227	0
Subset	proximal	None	None	e	91	220	0
Superset	precise	NewCocominMedian	None	lc	91	220	0
Superset	precise	NewCocominNative	None	lc	91	220	0
Superset	precise	None	None	cocomin1	91	220	0

Table 3.7: Top 10 methods for subset 3 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	NewCocominMedian	DynamicLocomoMean	lc	47	264	0
Superset	precise	NewCocominNative	DynamicLocomoMean	lc	47	264	0
Superset	precise	OldCocomin	DynamicLocomoMean	lc	47	264	0
Superset	precise	None	DynamicLocomoMean	lc	49	262	0
Superset	precise	NewCocominMedian	DynamicLocomoMedian	lc	51	260	0
Superset	precise	NewCocominNative	DynamicLocomoMedian	lc	51	260	0
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	51	260	0
Superset	precise	None	DynamicLocomoMedian	lc	61	250	0
Subset	precise	OldCocomin	StaticLocomo	lc	67	244	0
Subset	precise	NewCocominMedian	StaticLocomo	lc	77	234	0

Table 3.8: Top 10 methods for subset 4 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	None	None	e	18	293	0
Superset	proximal	None	None	e	18	293	0
Subset	precise	None	None	sd	20	291	0
Superset	precise	None	None	sd	20	291	0
Subset	precise	None	StaticLocomo	lc	25	286	0
Subset	precise	None	None	lc	44	267	0
Superset	precise	NewCocominNative	None	lc	50	261	0
Superset	precise	None	None	cocomin1	50	261	0
Superset	precise	None	None	cocomin2	50	261	0
Superset	precise	None	None	cocomost	50	261	0

Table 3.9: Top 10 methods for subset 5 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	StaticLocomo	lc	69	242	0
Subset	proximal	NewCocominMedian	StaticLocomo	lc	69	242	0
Subset	proximal	None	StaticLocomo	lc	69	242	0
Subset	precise	NewCocominNative	StaticLocomo	lc	70	241	0
Subset	precise	OldCocomin	StaticLocomo	lc	70	241	0
Subset	proximal	None	None	lc	70	241	0
Subset	precise	None	None	lc	72	239	0
Subset	proximal	NewCocominNative	StaticLocomo	lc	72	239	0
Subset	proximal	NewCocominMedian	None	lc	75	236	0
Subset	proximal	OldCocomin	StaticLocomo	lc	75	236	0

Table 3.10: Top 10 methods for subset 6 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	StaticLocomo	lc	9	302	0
Subset	precise	None	None	lc	10	301	0
Subset	precise	None	None	org	10	301	0
Superset	precise	None	None	org	10	301	0
Subset	precise	None	DynamicLocomoMedian	lc	16	295	0
Subset	proximal	None	None	lc	43	268	0
Subset	precise	None	DynamicLocomoMean	lc	46	265	0
Subset	proximal	None	None	org	49	262	0
Superset	proximal	None	None	org	49	262	0
Subset	precise	NewCocominMedian	DynamicLocomoMedian	lc	55	256	0

Table 3.11: Top 10 methods for subset 7 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	OldCocomin	DynamicLocomoMean	lc	182	129	0
Subset	precise	OldCocomin	StaticLocomo	lc	184	127	0
Superset	precise	OldCocomin	StaticLocomo	lc	184	127	0
Subset	precise	OldCocomin	DynamicLocomoMean	lc	232	79	0
Subset	precise	None	None	cocomin1	235	76	0
Subset	precise	OldCocomin	None	lc	235	76	0
Superset	precise	None	None	cocomin1	235	76	0
Superset	precise	OldCocomin	None	lc	235	76	0
Subset	precise	NewCocominMedian	StaticLocomo	lc	247	64	0
Subset	precise	None	None	lc	249	62	0

Table 3.12: Top 10 methods for subset 8 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	147	164	0
Superset	precise	OldCocomin	DynamicLocomoMean	lc	159	152	0
Superset	proximal	LocalWrapper	None	m5p	170	141	0
Superset	precise	None	None	cocomin1	184	127	0
Superset	precise	OldCocomin	None	lc	184	127	0
Superset	proximal	OldCocomin	None	lsr	187	124	0
Superset	proximal	OldCocomin	None	m5p	196	115	0
Superset	precise	OldCocomin	StaticLocomo	lc	198	113	0
Superset	precise	None	None	cocomost	201	110	0
Superset	proximal	None	None	m5p	201	110	0

Table 3.13: Top 10 methods for subset 9 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	None	None	e	5	306	0
Superset	proximal	None	None	e	5	306	0
Subset	precise	None	None	e	32	279	0
Superset	precise	None	None	e	32	279	0
Subset	proximal	None	None	org	154	157	0
Superset	proximal	None	None	org	154	157	0
Subset	proximal	OldCocomin	StaticLocomo	lc	87	222	2
Superset	proximal	NewCocominMedian	StaticLocomo	lc	105	204	2
Superset	proximal	None	None	cocomost	127	182	2
Superset	proximal	NewCocominMedian	DynamicLocomoMean	lc	131	178	2

Table 3.14: Top 10 methods for subset 10 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	sd	57	254	0
Superset	precise	None	None	sd	57	254	0
Subset	precise	None	None	lc	68	243	0
Subset	proximal	None	None	lc	80	231	0
Subset	precise	None	DynamicLocomoMean	lc	86	225	0
Subset	proximal	NewCocominMedian	None	lc	87	224	0
Subset	proximal	NewCocominMedian	DynamicLocomoMedian	lc	89	222	0
Subset	proximal	NewCocominNative	None	lc	90	221	0
Subset	proximal	None	None	cocomin2	90	221	0
Superset	precise	None	None	lc	90	221	0

Table 3.15: Top 10 methods for subset 11 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	NewCocominMedian	None	m5p	75	236	0
Superset	precise	LSRWrapper	None	lsr	87	224	0
Superset	precise	LSRWrapper	None	m5p	87	224	0
Superset	precise	NewCocominMedian	None	lsr	92	219	0
Superset	proximal	NewCocominNative	None	lsr	95	216	0
Superset	precise	M5PWrapper	None	lsr	111	200	0
Superset	precise	M5PWrapper	None	m5p	111	200	0
Superset	proximal	NewCocominNative	None	m5p	111	200	0
Superset	precise	OldCocomin	None	lsr	114	197	0
Superset	precise	OldCocomin	None	m5p	114	197	0

Table 3.16: Top 10 methods for subset 12 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	OldCocomin	None	lsr	68	243	0
Subset	proximal	M5PWrapper	None	lsr	76	235	0
Subset	proximal	NewCocominNative	None	lsr	77	234	0
Subset	precise	OldCocomin	None	lsr	78	233	0
Subset	proximal	NewCocominMedian	None	lsr	79	232	0
Subset	proximal	LSRWrapper	None	lsr	80	231	0
Subset	precise	NewCocominNative	None	lsr	83	228	0
Subset	proximal	None	None	lsr	87	224	0
Subset	proximal	LSRWrapper	None	m5p	90	221	0
Subset	proximal	M5PWrapper	None	m5p	103	208	0

Table 3.17: Top 10 methods for subset 13 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	None	None	lsr	62	249	0
Superset	proximal	None	None	lsr	76	235	0
Subset	precise	NewCocominMedian	StaticLocomo	lc	91	220	0
Subset	precise	NewCocominNative	StaticLocomo	lc	94	217	0
Superset	proximal	NewCocominNative	None	lsr	94	217	0
Subset	precise	None	StaticLocomo	lc	99	212	0
Superset	precise	NewCocominMedian	None	lsr	105	206	0
Superset	precise	NewCocominNative	None	lsr	105	206	0
Superset	precise	None	None	m5p	105	206	0
Superset	proximal	NewCocominMedian	None	lsr	107	204	0

Table 3.18: Top 10 methods for subset 14 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	NewCocominNative	None	lsr	54	257	0
Subset	precise	NewCocominMedian	None	lsr	100	211	0
Subset	precise	NewCocominNative	None	lsr	105	206	0
Subset	proximal	M5PWrapper	None	m5p	106	205	0
Subset	proximal	M5PWrapper	None	lsr	112	199	0
Subset	proximal	LSRWrapper	None	lsr	126	185	0
Subset	precise	OldCocomin	None	lsr	127	184	0
Subset	proximal	NewCocominMedian	None	lsr	132	179	0
Subset	proximal	OldCocomin	None	lsr	139	172	0
Subset	precise	OldCocomin	DynamicLocomoMean	lc	147	164	0

Table 3.19: Top 10 methods for subset 15 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	None	None	e	24	287	0
Superset	proximal	None	None	e	24	287	0
Superset	precise	None	DynamicLocomoMedian	lc	34	277	0
Subset	proximal	NewCocominMedian	None	lc	40	271	0
Subset	proximal	NewCocominMedian	StaticLocomo	lc	40	271	0
Subset	proximal	None	None	cocomin1	42	269	0
Subset	proximal	OldCocomin	None	lc	42	269	0
Subset	proximal	OldCocomin	StaticLocomo	lc	42	269	0
Subset	precise	NewCocominNative	None	lc	43	268	0
Subset	precise	None	None	cocomin2	43	268	0

Table 3.20: Top 10 methods for subset 16 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	proximal	NewCocominNative	None	lsr	95	216	0
Superset	proximal	NewCocominNative	None	m5p	117	194	0
Superset	precise	None	None	lsr	170	141	0
Superset	proximal	NewCocominMedian	None	m5p	171	140	0
Superset	proximal	None	None	lsr	171	140	0
Subset	precise	NewCocominMedian	None	lc	172	139	0
Subset	precise	LocalWrapper	None	lc	174	137	0
Superset	precise	None	None	m5p	177	134	0
Superset	precise	M5PWrapper	None	lsr	179	132	0
Superset	precise	M5PWrapper	None	m5p	179	132	0

Table 3.21: Top 10 methods for subset 17 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	M5PWrapper	None	m5p	101	210	0
Subset	precise	M5PWrapper	None	m5p	129	182	0
Subset	proximal	LSRWrapper	None	lsr	136	175	0
Subset	proximal	LSRWrapper	None	m5p	154	157	0
Subset	proximal	M5PWrapper	None	lsr	161	150	0
Subset	precise	LSRWrapper	None	m5p	175	136	0
Subset	precise	M5PWrapper	None	lsr	177	134	0
Subset	precise	NewCocominMedian	None	lsr	179	132	0
Subset	precise	OldCocomin	None	lsr	180	131	0
Superset	proximal	NewCocominNative	None	lsr	183	128	0

Table 3.22: Top 10 methods for subset 18 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	None	None	e	52	259	0
Superset	proximal	None	None	e	52	259	0
Subset	precise	None	None	cocomost	85	226	0
Superset	proximal	None	None	cocomin1	90	221	0
Superset	proximal	OldCocomin	None	lc	90	221	0
Subset	precise	None	None	e	105	206	0
Superset	precise	None	None	e	105	206	0
Superset	proximal	None	None	cocomost	105	206	0
Subset	precise	None	None	cocomin1	106	205	0
Subset	precise	OldCocomin	None	lc	106	205	0

Table 3.23: Top 10 methods for subset 19 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	e	59	252	0
Superset	precise	None	None	e	59	252	0
Subset	precise	None	StaticLocomo	lc	62	249	0
Superset	precise	None	StaticLocomo	lc	62	249	0
Subset	precise	NewCocominNative	StaticLocomo	lc	63	248	0
Superset	precise	NewCocominNative	StaticLocomo	lc	63	248	0
Subset	precise	OldCocomin	StaticLocomo	lc	64	247	0
Superset	precise	NewCocominMedian	StaticLocomo	lc	64	247	0
Superset	precise	OldCocomin	StaticLocomo	lc	64	247	0
Subset	precise	NewCocominMedian	StaticLocomo	lc	68	243	0

Table 3.24: Top 10 methods for subset 1 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	e	21	290	0
Superset	precise	None	None	e	21	290	0
Subset	proximal	None	None	e	25	286	0
Superset	proximal	None	None	e	25	286	0
Superset	precise	None	None	lc	33	278	0
Subset	precise	None	None	sd	45	266	0
Superset	precise	None	None	sd	45	266	0
Subset	precise	None	None	lc	63	248	0
Superset	precise	NewCocominMedian	None	lc	69	242	0
Superset	precise	NewCocominMedian	None	lsr	71	240	0

Table 3.25: Top 10 methods for subset 2 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	e	6	305	0
Superset	precise	None	None	e	6	305	0
Subset	precise	None	None	lc	30	281	0
Subset	precise	None	StaticLocomo	lc	30	281	0
Subset	precise	None	DynamicLocomoMean	lc	51	260	0
Subset	precise	None	DynamicLocomoMedian	lc	56	255	0
Superset	precise	None	DynamicLocomoMean	lc	71	238	2
Superset	precise	NewCocominMedian	DynamicLocomoMean	lc	79	230	2
Superset	precise	NewCocominNative	DynamicLocomoMean	lc	79	230	2
Superset	precise	OldCocomin	DynamicLocomoMean	lc	79	230	2

Table 3.26: Top 10 methods for subset 3 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	NewCocominMedian	DynamicLocomoMean	lc	13	298	0
Superset	precise	NewCocominMedian	DynamicLocomoMedian	lc	13	298	0
Superset	precise	NewCocominNative	DynamicLocomoMean	lc	13	298	0
Superset	precise	NewCocominNative	DynamicLocomoMedian	lc	13	298	0
Superset	precise	OldCocomin	DynamicLocomoMean	lc	13	298	0
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	13	298	0
Superset	precise	None	DynamicLocomoMean	lc	14	297	0
Superset	precise	None	DynamicLocomoMedian	lc	14	297	0
Subset	precise	None	None	cocomin1	47	264	0
Subset	precise	None	None	cocomost	47	264	0

Table 3.27: Top 10 methods for subset 4 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	StaticLocomo	lc	21	290	0
Subset	precise	None	None	lc	22	289	0
Subset	proximal	None	None	e	41	270	0
Superset	proximal	None	None	e	41	270	0
Superset	precise	NewCocominNative	DynamicLocomoMedian	lc	70	241	0
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	70	241	0
Subset	precise	None	DynamicLocomoMean	lc	73	238	0
Subset	precise	None	None	org	75	236	0
Subset	proximal	None	None	lc	75	236	0
Superset	precise	None	None	org	75	236	0

Table 3.28: Top 10 methods for subset 5 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	StaticLocomo	lc	21	290	0
Subset	precise	None	None	lc	32	279	0
Subset	proximal	None	StaticLocomo	lc	39	272	0
Subset	precise	None	DynamicLocomoMean	lc	40	271	0
Subset	precise	None	None	e	40	271	0
Subset	proximal	None	None	lc	40	271	0
Superset	precise	None	None	e	40	271	0
Subset	proximal	NewCocominMedian	None	lc	46	265	0
Subset	proximal	NewCocominNative	None	lc	53	258	0
Subset	proximal	None	None	cocomin2	53	258	0

Table 3.29: Top 10 methods for subset 6 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	org	6	305	0
Superset	precise	None	None	org	6	305	0
Subset	precise	None	None	lc	12	299	0
Subset	precise	None	StaticLocomo	lc	12	299	0
Subset	precise	None	DynamicLocomoMedian	lc	20	291	0
Subset	proximal	None	None	org	25	286	0
Superset	proximal	None	None	org	25	286	0
Subset	precise	None	None	sd	48	261	2
Superset	precise	None	None	sd	48	261	2
Subset	proximal	None	None	lc	54	255	2

Table 3.30: Top 10 methods for subset 7 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	cocomin1	65	246	0
Subset	precise	OldCocomin	None	lc	65	246	0
Superset	precise	None	None	cocomin1	65	246	0
Superset	precise	OldCocomin	None	lc	65	246	0
Subset	precise	None	None	e	100	211	0
Superset	precise	None	None	e	100	211	0
Subset	precise	OldCocomin	DynamicLocomoMedian	lc	110	201	0
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	110	201	0
Subset	precise	OldCocomin	StaticLocomo	lc	123	188	0
Superset	precise	OldCocomin	StaticLocomo	lc	123	188	0

Table 3.31: Top 10 methods for subset 8 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	None	None	cocomin1	51	260	0
Superset	precise	OldCocomin	None	lc	51	260	0
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	58	253	0
Superset	proximal	LocalWrapper	None	m5p	71	240	0
Superset	precise	None	None	cocomost	76	235	0
Superset	proximal	None	None	cocomost	78	233	0
Superset	proximal	OldCocomin	None	lsr	82	229	0
Superset	precise	LocalWrapper	None	lc	88	223	0
Superset	proximal	NewCocominMedian	None	lc	89	222	0
Superset	proximal	NewCocominMedian	None	lsr	89	222	0

Table 3.32: Top 10 methods for subset 9 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	proximal	NewCocominMedian	None	lc	18	293	0
Superset	proximal	None	None	cocomost	19	292	0
Subset	proximal	None	None	e	22	289	0
Superset	proximal	None	None	e	22	289	0
Superset	proximal	NewCocominMedian	StaticLocomo	lc	25	286	0
Superset	proximal	None	None	cocomin1	25	286	0
Superset	proximal	OldCocomin	None	lc	25	286	0
Superset	proximal	NewCocominMedian	DynamicLocomoMean	lc	28	283	0
Superset	precise	None	None	cocomin1	37	274	0
Superset	precise	OldCocomin	None	lc	37	274	0

Table 3.33: Top 10 methods for subset 10 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	lc	23	288	0
Subset	precise	None	DynamicLocomoMean	lc	32	279	0
Subset	precise	None	StaticLocomo	lc	40	271	0
Superset	precise	None	None	lc	47	264	0
Subset	proximal	None	None	lc	49	262	0
Subset	precise	None	DynamicLocomoMedian	lc	61	250	0
Subset	proximal	NewCocominNative	None	lc	62	249	0
Subset	proximal	None	None	cocomin2	62	249	0
Subset	proximal	NewCocominMedian	DynamicLocomoMedian	lc	64	247	0
Subset	proximal	NewCocominMedian	None	lc	64	247	0

Table 3.34: Top 10 methods for subset 11 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	proximal	NewCocominNative	None	lsr	79	232	0
Superset	precise	NewCocominMedian	None	m5p	81	230	0
Subset	precise	LocalWrapper	None	lc	83	228	0
Superset	proximal	NewCocominNative	None	m5p	86	225	0
Superset	precise	LSRWrapper	None	lsr	87	224	0
Superset	precise	LSRWrapper	None	m5p	87	224	0
Superset	proximal	NewCocominMedian	None	lsr	87	224	0
Superset	precise	OldCocomin	None	lsr	89	222	0
Superset	precise	OldCocomin	None	m5p	89	222	0
Superset	precise	NewCocominMedian	None	lsr	90	221	0

Table 3.35: Top 10 methods for subset 12 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	OldCocomin	None	lsr	35	276	0
Subset	proximal	NewCocominMedian	None	lsr	44	267	0
Subset	proximal	NewCocominNative	None	lsr	44	267	0
Subset	proximal	None	None	lsr	49	262	0
Subset	precise	OldCocomin	None	lsr	53	258	0
Subset	precise	NewCocominNative	None	lsr	56	255	0
Subset	proximal	LSRWrapper	None	lsr	56	255	0
Subset	proximal	M5PWrapper	None	lsr	56	255	0
Subset	proximal	LSRWrapper	None	m5p	57	254	0
Subset	proximal	NewCocominNative	None	m5p	67	244	0

Table 3.36: Top 10 methods for subset 13 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	StaticLocomo	lc	25	286	0
Subset	precise	OldCocomin	StaticLocomo	lc	25	286	0
Subset	precise	NewCocominMedian	StaticLocomo	lc	28	283	0
Subset	precise	NewCocominNative	StaticLocomo	lc	33	278	0
Subset	precise	None	None	cocomin1	47	264	0
Subset	precise	OldCocomin	None	lc	47	264	0
Subset	precise	None	None	lc	54	257	0
Subset	precise	NewCocominMedian	None	lc	56	255	0
Subset	precise	None	None	cocomost	57	254	0
Subset	precise	OldCocomin	DynamicLocomoMedian	lc	65	246	0

Table 3.37: Top 10 methods for subset 14 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	NewCocominNative	None	lsr	29	282	0
Subset	precise	NewCocominNative	None	lsr	56	255	0
Subset	precise	NewCocominMedian	None	lsr	57	254	0
Subset	precise	OldCocomin	None	lsr	69	242	0
Subset	proximal	M5PWrapper	None	m5p	72	239	0
Subset	proximal	NewCocominMedian	None	lsr	74	237	0
Subset	proximal	M5PWrapper	None	lsr	80	231	0
Subset	proximal	LSRWrapper	None	lsr	81	230	0
Subset	precise	OldCocomin	DynamicLocomoMedian	lc	87	224	0
Subset	proximal	OldCocomin	None	lsr	90	221	0

Table 3.38: Top 10 methods for subset 15 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	None	DynamicLocomoMedian	lc	22	289	0
Subset	proximal	NewCocominMedian	None	lc	38	273	0
Subset	proximal	NewCocominMedian	StaticLocomo	lc	39	272	0
Subset	proximal	None	None	cocomin1	44	267	0
Subset	proximal	OldCocomin	None	lc	44	267	0
Subset	proximal	OldCocomin	StaticLocomo	lc	44	267	0
Subset	precise	NewCocominNative	None	lc	48	263	0
Subset	precise	None	None	cocomin2	48	263	0
Subset	proximal	None	None	cocomost	48	263	0
Subset	proximal	None	None	e	48	263	0

Table 3.39: Top 10 methods for subset 16 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	None	None	e	92	219	0
Superset	proximal	None	None	e	92	219	0
Superset	proximal	NewCocominNative	None	lsr	94	217	0
Subset	precise	LSRWrapperNoLog	None	m5p	97	214	0
Subset	precise	M5PWrapperNoLog	None	m5p	103	208	0
Subset	precise	LocalWrapper	None	lc	107	204	0
Superset	proximal	NewCocominNative	None	m5p	110	201	0
Subset	precise	M5PWrapper	None	lsr	111	200	0
Superset	proximal	None	None	lsr	112	199	0
Subset	precise	LSRWrapperNoLog	None	lsr	113	198	0

Table 3.40: Top 10 methods for subset 17 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	M5PWrapper	None	m5p	30	281	0
Subset	precise	M5PWrapper	None	m5p	51	260	0
Subset	proximal	LSRWrapper	None	lsr	51	260	0
Subset	proximal	M5PWrapper	None	lsr	53	258	0
Subset	proximal	LSRWrapper	None	m5p	61	250	0
Subset	precise	NewCocominMedian	None	lsr	71	240	0
Subset	precise	LSRWrapper	None	m5p	72	239	0
Subset	precise	NewCocominNative	None	lsr	72	239	0
Subset	precise	OldCocomin	None	lsr	73	238	0
Subset	precise	LSRWrapper	None	lsr	76	235	0

Table 3.41: Top 10 methods for subset 18 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	proximal	None	None	cocomin1	44	267	0
Superset	proximal	OldCocomin	None	lc	44	267	0
Superset	proximal	None	None	cocomost	51	260	0
Subset	precise	None	None	cocomin1	52	259	0
Subset	precise	OldCocomin	None	lc	52	259	0
Subset	precise	None	None	cocomost	59	252	0
Superset	proximal	NewCocominMedian	None	lc	61	250	0
Subset	proximal	None	None	e	67	244	0
Superset	proximal	None	None	e	67	244	0
Superset	precise	None	None	cocomin1	68	243	0

Table 3.42: Top 10 methods for subset 19 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	NewCocominNative	StaticLocomo	lc	51	260	0
Superset	precise	NewCocominNative	StaticLocomo	lc	51	260	0
Subset	precise	None	StaticLocomo	lc	53	258	0
Subset	precise	OldCocomin	StaticLocomo	lc	53	258	0
Superset	precise	NewCocominMedian	StaticLocomo	lc	53	258	0
Superset	precise	None	StaticLocomo	lc	53	258	0
Superset	precise	OldCocomin	StaticLocomo	lc	53	258	0
Subset	precise	NewCocominMedian	StaticLocomo	lc	59	252	0
Subset	precise	None	DynamicLocomoMedian	lc	59	252	0
Subset	precise	None	None	sd	59	252	0

Table 3.43: Top 10 methods for subset 1 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	NewCocominMedian	None	lsr	49	262	0
Superset	precise	NewCocominNative	None	lsr	49	262	0
Superset	precise	OldCocomin	None	lsr	49	262	0
Superset	precise	None	None	lsr	51	260	0
Superset	precise	None	StaticLocomo	lc	51	260	0
Superset	precise	None	None	lc	52	259	0
Subset	precise	None	None	sd	53	258	0
Superset	precise	None	None	sd	53	258	0
Superset	precise	NewCocominMedian	StaticLocomo	lc	61	250	0
Superset	precise	NewCocominNative	StaticLocomo	lc	61	250	0

Table 3.44: Top 10 methods for subset 2 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	e	22	289	0
Superset	precise	None	None	e	22	289	0
Subset	precise	None	None	lc	28	283	0
Subset	precise	None	StaticLocomo	lc	28	283	0
Superset	precise	None	StaticLocomo	lc	28	283	0
Superset	precise	NewCocominMedian	StaticLocomo	lc	29	282	0
Superset	precise	NewCocominNative	StaticLocomo	lc	29	282	0
Superset	precise	OldCocomin	StaticLocomo	lc	29	282	0
Subset	proximal	None	None	e	31	280	0
Superset	proximal	None	None	e	31	280	0

Table 3.45: Top 10 methods for subset 3 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	NewCocominMedian	DynamicLocomoMean	lc	9	302	0
Superset	precise	NewCocominNative	DynamicLocomoMean	lc	9	302	0
Superset	precise	None	DynamicLocomoMean	lc	9	302	0
Superset	precise	OldCocomin	DynamicLocomoMean	lc	9	302	0
Superset	precise	NewCocominMedian	DynamicLocomoMedian	lc	12	299	0
Superset	precise	NewCocominNative	DynamicLocomoMedian	lc	12	299	0
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	12	299	0
Superset	precise	None	DynamicLocomoMedian	lc	15	296	0
Subset	precise	None	None	org	35	276	0
Superset	precise	None	None	org	35	276	0

Table 3.46: Top 10 methods for subset 4 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	lc	21	290	0
Subset	precise	None	StaticLocomo	lc	23	288	0
Subset	proximal	None	None	e	25	286	0
Superset	proximal	None	None	e	25	286	0
Subset	precise	None	None	org	29	282	0
Superset	precise	None	None	org	29	282	0
Superset	precise	None	None	lc	33	278	0
Superset	precise	NewCocominNative	None	lc	35	276	0
Superset	precise	None	None	cocomin1	35	276	0
Superset	precise	None	None	cocomin2	35	276	0

Table 3.47: Top 10 methods for subset 5 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	StaticLocomo	lc	21	290	0
Subset	precise	None	None	e	22	289	0
Superset	precise	None	None	e	22	289	0
Subset	precise	None	None	lc	37	274	0
Subset	proximal	None	None	e	51	260	0
Superset	proximal	None	None	e	51	260	0
Subset	proximal	None	StaticLocomo	lc	54	257	0
Subset	proximal	None	None	lc	66	245	0
Subset	precise	None	DynamicLocomoMean	lc	71	240	0
Subset	proximal	NewCocominMedian	None	lc	77	234	0

Table 3.48: Top 10 methods for subset 6 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	org	7	304	0
Superset	precise	None	None	org	7	304	0
Subset	precise	None	None	lc	8	303	0
Subset	precise	None	StaticLocomo	lc	8	303	0
Subset	precise	None	DynamicLocomoMedian	lc	12	299	0
Subset	precise	None	DynamicLocomoMean	lc	33	278	0
Subset	proximal	None	None	org	34	277	0
Superset	proximal	None	None	org	34	277	0
Subset	proximal	None	None	lc	44	265	2
Subset	precise	None	None	cocomin1	43	264	4

Table 3.49: Top 10 methods for subset 7 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	OldCocomin	StaticLocomo	lc	116	195	0
Superset	precise	OldCocomin	StaticLocomo	lc	116	195	0
Superset	precise	OldCocomin	DynamicLocomoMean	lc	118	193	0
Subset	precise	OldCocomin	None	lc	129	182	0
Superset	precise	None	None	cocomin1	129	182	0
Superset	precise	OldCocomin	None	lc	129	182	0
Subset	precise	None	None	cocomin1	130	181	0
Subset	precise	OldCocomin	DynamicLocomoMedian	lc	132	179	0
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	132	179	0
Subset	precise	None	StaticLocomo	lc	137	174	0

Table 3.50: Top 10 methods for subset 8 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	OldCocomin	DynamicLocomoMean	lc	51	260	0
Superset	proximal	LSRWrapper	None	lsr	70	241	0
Superset	proximal	LSRWrapper	None	m5p	70	241	0
Superset	proximal	M5PWrapper	None	lsr	96	215	0
Superset	proximal	M5PWrapper	None	m5p	96	215	0
Superset	proximal	NewCocominMedian	None	lsr	104	207	0
Superset	precise	OldCocomin	StaticLocomo	lc	114	197	0
Superset	proximal	NewCocominNative	None	m5p	118	193	0
Superset	proximal	OldCocomin	None	lsr	122	189	0
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	125	186	0

Table 3.51: Top 10 methods for subset 9 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	proximal	NewCocominMedian	StaticLocomo	lc	46	265	0
Superset	proximal	NewCocominMedian	DynamicLocomoMean	lc	49	262	0
Subset	precise	None	None	e	57	254	0
Superset	precise	None	None	e	57	254	0
Superset	proximal	NewCocominNative	StaticLocomo	lc	83	228	0
Superset	proximal	OldCocomin	DynamicLocomoMedian	lc	97	214	0
Superset	proximal	NewCocominMedian	None	lc	98	213	0
Superset	proximal	NewCocominNative	DynamicLocomoMean	lc	115	196	0
Superset	precise	None	None	cocomin1	118	193	0
Superset	precise	OldCocomin	None	lc	118	193	0

Table 3.52: Top 10 methods for subset 10 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	lc	17	294	0
Subset	precise	None	DynamicLocomoMean	lc	27	284	0
Subset	proximal	None	None	lc	34	277	0
Subset	precise	None	DynamicLocomoMedian	lc	50	261	0
Subset	proximal	NewCocominNative	None	lc	57	254	0
Subset	proximal	None	None	cocomin2	57	254	0
Subset	proximal	NewCocominMedian	None	lc	58	253	0
Subset	proximal	NewCocominMedian	DynamicLocomoMedian	lc	60	251	0
Superset	precise	None	None	lc	67	244	0
Subset	proximal	NewCocominMedian	DynamicLocomoMean	lc	69	242	0

Table 3.53: Top 10 methods for subset 11 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	LSRWrapper	None	lsr	32	279	0
Superset	precise	LSRWrapper	None	m5p	32	279	0
Superset	precise	NewCocominMedian	None	lsr	35	276	0
Superset	proximal	NewCocominNative	None	lsr	51	260	0
Superset	proximal	NewCocominNative	None	m5p	52	259	0
Superset	proximal	NewCocominMedian	None	m5p	54	257	0
Superset	proximal	NewCocominMedian	None	lsr	56	255	0
Superset	proximal	OldCocomin	None	m5p	58	253	0
Superset	proximal	OldCocomin	None	lsr	66	245	0
Superset	precise	NewCocominNative	None	m5p	71	240	0

Table 3.54: Top 10 methods for subset 12 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	OldCocomin	None	lsr	33	278	0
Subset	proximal	NewCocominNative	None	lsr	34	277	0
Subset	proximal	NewCocominMedian	None	lsr	40	271	0
Subset	proximal	None	None	lsr	46	265	0
Subset	precise	NewCocominNative	None	lsr	50	261	0
Subset	precise	OldCocomin	None	lsr	51	260	0
Subset	proximal	LSRWrapper	None	lsr	53	258	0
Subset	proximal	M5PWrapper	None	lsr	53	258	0
Subset	proximal	LSRWrapper	None	m5p	55	256	0
Subset	proximal	NewCocominNative	None	m5p	60	251	0

Table 3.55: Top 10 methods for subset 13 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	e	7	304	0
Superset	precise	None	None	e	7	304	0
Superset	precise	None	None	lsr	53	258	0
Superset	proximal	None	None	lsr	61	250	0
Subset	precise	None	None	cocomin1	71	240	0
Subset	precise	OldCocomin	None	lc	71	240	0
Subset	precise	None	None	sd	61	249	1
Superset	precise	None	None	sd	61	249	1
Subset	precise	None	StaticLocomo	lc	48	261	2
Subset	precise	OldCocomin	StaticLocomo	lc	50	259	2

Table 3.56: Top 10 methods for subset 14 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	NewCocominNative	None	lsr	26	285	0
Subset	precise	NewCocominNative	None	lsr	54	257	0
Subset	precise	NewCocominMedian	None	lsr	55	256	0
Subset	proximal	NewCocominMedian	None	lsr	74	237	0
Subset	proximal	M5PWrapper	None	m5p	75	236	0
Subset	precise	OldCocomin	None	lsr	79	232	0
Subset	proximal	LSRWrapper	None	lsr	80	231	0
Subset	proximal	M5PWrapper	None	lsr	81	230	0
Subset	proximal	NewCocominNative	None	m5p	84	227	0
Subset	proximal	OldCocomin	None	lsr	86	225	0

Table 3.57: Top 10 methods for subset 15 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	precise	None	DynamicLocomoMedian	lc	29	282	0
Subset	proximal	NewCocominMedian	StaticLocomo	lc	39	272	0
Subset	proximal	NewCocominMedian	None	lc	40	271	0
Subset	proximal	None	None	cocomin1	40	271	0
Subset	proximal	OldCocomin	None	lc	40	271	0
Subset	proximal	OldCocomin	StaticLocomo	lc	40	271	0
Subset	precise	NewCocominMedian	None	lc	42	269	0
Subset	precise	NewCocominNative	None	lc	42	269	0
Subset	precise	None	None	cocomin2	42	269	0
Subset	proximal	NewCocominNative	None	lc	42	269	0

Table 3.58: Top 10 methods for subset 16 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Superset	proximal	NewCocominNative	None	lsr	70	241	0
Superset	precise	M5PWrapper	None	lsr	78	233	0
Superset	precise	M5PWrapper	None	m5p	78	233	0
Superset	proximal	NewCocominNative	None	m5p	83	228	0
Superset	precise	LSRWrapper	None	lsr	84	227	0
Superset	precise	LSRWrapper	None	m5p	84	227	0
Superset	proximal	NewCocominMedian	None	m5p	93	218	0
Superset	precise	NewCocominMedian	None	m5p	94	217	0
Superset	precise	None	None	lsr	95	216	0
Superset	precise	None	None	m5p	95	216	0

Table 3.59: Top 10 methods for subset 17 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	proximal	M5PWrapper	None	m5p	41	270	0
Subset	precise	M5PWrapper	None	m5p	49	262	0
Subset	proximal	LSRWrapper	None	lsr	55	256	0
Subset	proximal	M5PWrapper	None	lsr	58	253	0
Subset	proximal	LSRWrapper	None	m5p	60	251	0
Subset	precise	M5PWrapper	None	lsr	64	247	0
Superset	proximal	NewCocominNative	None	lsr	64	247	0
Subset	precise	LSRWrapper	None	m5p	70	241	0
Superset	proximal	NewCocominNative	None	m5p	71	240	0
Superset	proximal	None	None	lsr	72	239	0

Table 3.60: Top 10 methods for subset 18 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	e	46	265	0
Superset	precise	None	None	e	46	265	0
Superset	precise	None	None	lc	59	252	0
Subset	precise	None	None	lc	65	246	0
Subset	precise	None	StaticLocomo	lc	65	246	0
Superset	proximal	None	None	cocomin1	66	245	0
Superset	proximal	OldCocomin	None	lc	66	245	0
Subset	precise	None	None	cocomin1	68	243	0
Subset	precise	OldCocomin	None	lc	68	243	0
Superset	proximal	None	None	lc	68	243	0

Table 3.61: Top 10 methods for subset 19 using MRE evaluation criterion.

All Subsets And One Evaluation Criterion

Tables 3.5 through 3.23 show the results of running the MWU test using the AR evaluation criterion for each of the 19 subsets of *COC*81 and *NASA*93 data sets. The names of these data sets can be found from Table 3.1.

As expected, the top 10 methods for each subset is different from other subsets. There is no general trend as to which method performs better in each subset. For example, both superset and subset train types are present in the results. Also, both precise and proximal numeric estimation methods are present in the top 10 methods for these subsets. Moreover, column pruners, row pruners, and learners based on local calibration as well as other methods are represented in the results.

Nonetheless, this was anticipated. A single method cannot possibly be useful for all subset and data sets. If so, such a method would have been reported in the literature. However, no such method is known to exist at the time of writing the results of this study. Therefore, this study is in agreement with other studies in this respect. However, this work does not consider these results without any direction or decisive content. Although no single method is best, there are certain points that can be made about the top methods for the subsets used in this study. Furthermore, clear points can be made about methods not in the top reported list.

These findings for AR are as follows:

1. In 18 of 19 subsets, the top method either had a simple column pruning method (Cocomin) or required no column pruning method at all. This is a key finding since it simplifies and speeds up the effort estimation process.

2. In all subsets, the top method either used no row pruning method or used one based on local calibration.

3. In addition, the nearest neighbor method was not used in the top 10 performing methods. This and the previous point clearly state that row pruning without any domain knowledge,

which is the nearest neighbor method in this case, is actually less useful than not doing row pruning at all.

4. All the top methods in every subset used learners employing linear regression to learn a model. Local calibration and all methods based on that, as well as LSR and M5p are such methods. Once again the nearest neighbor method that does not use any of these learners fails. This re-emphasizes the need for methods that have some domain knowledge about the data (where it is assumed by methods such as local calibration that project effort is exponential on *KLOC*).

5. In 12 of 19 subsets, local calibration and learners based on that, including its standard modes were the top method's learner. LSR was used in the other 5 subsets and M5p was only used in 2 subsets. This clearly states that in 17 of the 19 subsets, model trees were not selected as the preferred learner and simpler linear regression learners performed better.

6. In all subsets, when column pruning using LSR or M5p as target learners is present, it logs the data. The absence of methods that do not log the data simplifies the search for the top performing methods in Section 3.6.5.

7. Although simple column pruners based on local calibration (such as Cocomin) are useful, the more thorough one (Local Wrapper) is never used in the top method. Hence, sophisticated column pruners can be removed when looking for the top performing methods in Section 3.6.5.

All Subsets And All Evaluation Criteria

The importance of these findings is that, they can be generalized to other evaluation criteria. The results of the MWU test based on the MER evaluation criterion are presented in Tables 3.24 through 3.42. Similarly, the results based on the MRE evaluation criterion are presented in Tables 3.43

through 3.61. The goal is to identify any similarities or differences in the results across these evaluation criteria for each individual subset.

- Finding 1 holds for both MER and MRE. In MER results, the top methods of 18 of 19 subsets used a simple column pruning method (Cocomin) or required no column pruning method at all. In MRE tables, 17 of the 19 subsets showed the same results, with 1 of the 2 other subset showing Cocomin very close to the top method.
- Finding 2 holds for all subsets evaluated using MER and MRE as well.
- Finding 3 holds for all subsets evaluated using MER and MRE as well. In addition, these two findings demonstrate the importance of using methods that have some domain knowledge about the data.
- Finding 4 holds for all subsets evaluated using MER and MRE as well.
- Finding 5 holds for MER and MRE as well. For MER, 15 of the subsets used local calibration and learners based on it and 3 used LSR. Only 1 used M5p, meaning that 18 of the 19 subsets used simple linear regression learners. For MRE, 13 used local calibration and learners based on it, 5 used LSR, and only 1 used M5p. Therefore, 18 of the 19 subsets used simpler linear regression learners again. It is even more interesting that in 15 of the 19 subsets, the top method's learner across all evaluation criteria was from the same type of learner (meaning that for all three criteria, it was either based on local calibration, or was one of the LSR or M5p learners). This further shows the agreement between different evaluation criteria.
- Finding 6 holds for all subsets evaluated using MER and MRE as well except for subset 17 evaluated using MER. However, a simple local calibration performs better in this subset. Therefore, the same point can be made about simplifying the search for top performing methods by removing methods that do not log the data.

- Finding 7 holds for all subsets when using MER and MRE as well. Sophisticated and complex column pruning methods are not required in general.

The agreement in the findings among the three evaluation criteria and across all 19 subsets proves the fact that different evaluation criteria show stable results (using the MWU test). This is an essential point in studying the effects of evaluation bias in effort estimation studies and is further discussed in Chapter 4.

3.6.2 Top 20 Methods For Each Data Set Across Three Evaluation Criteria

Although the results for the subsets of each data set are useful in finding the most appropriate methods for each subset, it is also useful to find the best methods applied to each data set based on the results for each individual subset of that data set. This is done by summing the number of ties, wins, and losses individually for each method across all subsets of a data set. In this section, the top 20 methods for each data set ($COC81$ and $NASA93$) are reported. This includes the top 6% of the methods that have the fewest number of total losses over all subsets of that data set (and most number of total wins in case of the same number of total losses). The justification for not showing more results is offered in Section 3.6.3.

The goal of presenting the results in this way is to find methods that can perform best across most, if not all, subsets of a data set. Similar to reporting the results in Section 3.6.1, these results are also reported for two data sets, $COC81$ and $NASA93$, across all three evaluation criteria, AR, MER, and MRE. Tables 3.62 through 3.67 contain these results. It should be noted that the total number of losses (as well as wins and ties) possible is $311 * 7 = 2177$ for $COC81$ and $311 * 12 = 3732$ for $NASA93$.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	DynamicLocomoMean	lc	743	1430	4
Subset	precise	None	DynamicLocomoMedian	lc	796	1377	4
Subset	precise	None	None	lc	750	1417	10
Subset	precise	NewCocominMedian	None	lc	922	1244	11
Subset	precise	None	StaticLocomo	lc	529	1631	17
Superset	precise	NewCocominNative	DynamicLocomoMedian	lc	868	1288	21
Superset	precise	NewCocominMedian	DynamicLocomoMedian	lc	892	1264	21
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	892	1264	21
Subset	precise	NewCocominMedian	DynamicLocomoMedian	lc	965	1190	22
Subset	precise	OldCocomin	DynamicLocomoMean	lc	903	1250	24
Subset	precise	None	None	cocomin1	925	1228	24
Subset	precise	OldCocomin	None	lc	925	1228	24
Subset	precise	NewCocominNative	DynamicLocomoMean	lc	938	1215	24
Subset	precise	NewCocominMedian	StaticLocomo	lc	685	1467	25
Subset	precise	NewCocominNative	None	lc	936	1216	25
Subset	precise	None	None	cocomin2	936	1216	25
Subset	precise	NewCocominNative	StaticLocomo	lc	682	1469	26
Subset	precise	NewCocominMedian	DynamicLocomoMean	lc	900	1250	27
Subset	precise	NewCocominNative	DynamicLocomoMedian	lc	963	1184	30
Subset	proximal	None	DynamicLocomoMean	lc	1143	1004	30

Table 3.62: Top 20 methods for *COC*81 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	NewCocominMedian	None	lc	2355	1373	4
Subset	precise	None	None	cocomost	2172	1555	5
Subset	proximal	None	None	e	2217	1510	5
Superset	proximal	None	None	e	2217	1510	5
Subset	precise	None	None	cocomin1	2154	1571	7
Subset	precise	OldCocomin	None	lc	2154	1571	7
Subset	precise	NewCocominNative	None	lc	2474	1249	9
Subset	precise	None	None	cocomin2	2474	1249	9
Superset	precise	None	None	lc	2350	1371	11
Subset	precise	OldCocomin	DynamicLocomoMedian	lc	2467	1253	12
Subset	precise	NewCocominNative	DynamicLocomoMedian	lc	2682	1037	13
Subset	precise	NewCocominMedian	DynamicLocomoMedian	lc	2693	1024	15
Superset	precise	OldCocomin	DynamicLocomoMean	lc	2472	1243	17
Subset	precise	OldCocomin	DynamicLocomoMean	lc	2437	1273	22
Subset	precise	None	None	lc	2525	1183	24
Superset	precise	None	DynamicLocomoMedian	lc	2659	1046	27
Superset	precise	None	DynamicLocomoMean	lc	2594	1106	32
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	2379	1318	35
Subset	precise	NewCocominNative	DynamicLocomoMean	lc	2679	1016	37
Superset	precise	None	None	cocomin1	2268	1423	41

Table 3.63: Top 20 methods for *NASA*93 using AR evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	lc	335	1829	13
Subset	precise	None	DynamicLocomoMean	lc	548	1608	21
Subset	precise	None	StaticLocomo	lc	281	1873	23
Subset	precise	None	None	cocomin1	555	1599	23
Subset	precise	OldCocomin	None	lc	555	1599	23
Subset	precise	NewCocominNative	None	lc	566	1587	24
Subset	precise	None	None	cocomin2	566	1587	24
Subset	precise	None	DynamicLocomoMedian	lc	562	1590	25
Subset	precise	NewCocominMedian	None	lc	568	1579	30
Subset	precise	None	None	cocomost	568	1577	32
Subset	precise	None	None	e	321	1817	39
Superset	precise	None	None	e	321	1817	39
Subset	precise	NewCocominMedian	StaticLocomo	lc	457	1649	71
Subset	precise	None	None	sd	497	1609	71
Superset	precise	None	None	sd	497	1609	71
Superset	precise	NewCocominMedian	DynamicLocomoMedian	lc	569	1535	73
Subset	precise	NewCocominMedian	DynamicLocomoMean	lc	621	1483	73
Superset	precise	NewCocominNative	DynamicLocomoMedian	lc	568	1535	74
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	570	1533	74
Subset	precise	OldCocomin	DynamicLocomoMean	lc	615	1486	76

Table 3.64: Top 20 methods for *COC*81 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	cocomin1	1388	2301	43
Subset	precise	OldCocomin	None	lc	1389	2300	43
Subset	precise	None	None	cocomost	1503	2181	48
Subset	proximal	None	None	e	1493	2181	58
Superset	proximal	None	None	e	1493	2181	58
Subset	precise	NewCocominMedian	None	lc	1702	1966	64
Subset	precise	NewCocominNative	None	lc	1829	1832	71
Subset	precise	None	None	cocomin2	1829	1832	71
Subset	proximal	None	None	cocomost	1777	1879	76
Subset	proximal	None	None	cocomin1	1827	1825	80
Subset	proximal	OldCocomin	None	lc	1827	1825	80
Subset	proximal	NewCocominMedian	None	lc	1986	1665	81
Subset	precise	None	None	e	726	2892	114
Superset	precise	None	None	e	726	2892	114
Subset	precise	None	None	lc	1737	1876	119
Subset	proximal	NewCocominNative	None	lc	2090	1519	123
Subset	proximal	None	None	cocomin2	2090	1519	123
Subset	precise	OldCocomin	DynamicLocomoMedian	lc	1789	1793	150
Superset	precise	None	None	lc	1632	1948	152
Subset	precise	NewCocominNative	DynamicLocomoMedian	lc	2131	1439	162

Table 3.65: Top 20 methods for *NASA*93 using MER evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	DynamicLocomoMean	lc	428	1739	10
Subset	precise	None	StaticLocomo	lc	260	1905	12
Subset	precise	None	None	lc	321	1841	15
Subset	precise	None	None	sd	348	1810	19
Superset	precise	None	None	sd	348	1810	19
Subset	precise	None	DynamicLocomoMedian	lc	436	1718	23
Superset	precise	NewCocominNative	DynamicLocomoMedian	lc	457	1687	33
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	462	1682	33
Superset	precise	NewCocominMedian	DynamicLocomoMedian	lc	472	1670	35
Superset	precise	NewCocominMedian	DynamicLocomoMean	lc	485	1645	47
Superset	precise	NewCocominNative	DynamicLocomoMean	lc	473	1656	48
Superset	precise	OldCocomin	DynamicLocomoMean	lc	473	1656	48
Subset	precise	NewCocominMedian	None	lc	566	1563	48
Superset	precise	None	StaticLocomo	lc	358	1770	49
Subset	precise	None	None	cocomost	585	1540	52
Superset	precise	None	DynamicLocomoMean	lc	473	1650	54
Superset	precise	None	DynamicLocomoMedian	lc	495	1626	56
Superset	precise	NewCocominNative	StaticLocomo	lc	362	1755	60
Superset	precise	OldCocomin	StaticLocomo	lc	364	1753	60
Superset	precise	NewCocominMedian	StaticLocomo	lc	365	1751	61

Table 3.66: Top 20 methods for *COC*81 using MRE evaluation criterion.

Train Type	Numeric Estimation	Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
Subset	precise	None	None	e	1558	2154	20
Superset	precise	None	None	e	1558	2154	20
Subset	precise	OldCocomin	None	lc	1438	2272	22
Subset	precise	None	None	cocomin1	1439	2271	22
Subset	precise	None	None	cocomost	1519	2187	26
Subset	precise	NewCocominMedian	None	lc	1751	1954	27
Subset	precise	NewCocominNative	None	lc	1886	1808	38
Subset	precise	None	None	cocomin2	1886	1808	38
Superset	precise	None	None	lc	1652	2022	58
Subset	precise	None	None	lc	1699	1962	71
Superset	precise	None	None	cocomin1	1538	2115	79
Superset	precise	OldCocomin	None	lc	1538	2115	79
Subset	precise	OldCocomin	DynamicLocomoMedian	lc	1795	1858	79
Superset	precise	OldCocomin	DynamicLocomoMean	lc	1715	1931	86
Superset	precise	OldCocomin	DynamicLocomoMedian	lc	1639	2003	90
Superset	precise	None	DynamicLocomoMean	lc	1931	1707	94
Superset	precise	None	DynamicLocomoMedian	lc	1991	1643	98
Subset	precise	NewCocominNative	DynamicLocomoMedian	lc	2140	1488	104
Subset	precise	OldCocomin	DynamicLocomoMean	lc	1709	1909	114
Superset	precise	None	StaticLocomo	lc	1344	2272	116

Table 3.67: Top 20 methods for *NASA*93 using MRE evaluation criterion.

Looking at the results for *COC*81 and *NASA*93 across all three evaluation criteria, these points can be made about the results in of Tables 3.62 through 3.67:

1. The only methods present in all 6 tables are based on the local calibration method. In every case, the learner is either the local calibration method or is based on local calibration, including the standard modes. This is an outstanding finding. This simply means that none of the more sophisticated column pruning methods (such as Wrappers that use LSR or M5p target learners) and also learners such as LSR and M5p are as useful as methods based on local calibration in effort estimation.

2. Although methods based on local calibration are the only top performers in these results, they do not require to be sophisticated. As an example, Local Wrapper, which uses a thorough search and the local calibration method, is not present in these results. Hence, it is adequate to use simple pre-processors and learners based on local calibration.

3. In addition to the absence of sophisticated methods, methods such as nearest neighbor that do not use the domain knowledge about the data are also absent from these results. Such methods represent the other end of the spectrum with their simplistic designs.

4. In all the results, the top two methods use the subset train type and the precise numeric estimation method. This can greatly simplify the search for the top performing methods in Section 3.6.5 by removing methods that use the superset train type and the proximal numeric estimation method.

5. In every data set and for all the evaluation criteria, the methods using dynamic Locomo always ranked higher than the methods using static Locomo. This shows that fixed neighborhoods cannot be as effective in row pruning as specific neighborhoods found for each data set. Therefore, although simpler methods are usually desired, row pruning requires a focused search of the instances to find the most appropriate neighborhood.

6. All the methods found in these results use linear regression (since they are all based on local calibration) and all log the data. This clarifies the need for logging the data.

7. Except in one case (Table 3.67) one or both of the top two methods use one or more of a column pruner, a row pruner, or a learner such as Cocomin that prunes columns automatically in training the model. Even in that one case, the next methods (which are very close to the top two) satisfy this finding. Therefore, pruning the data in effort estimation can be considered beneficial.

Similar to the case with the subsets, these findings across different data sets and evaluation criteria also show stability in the results. The stability seen across the data sets has a different nature from the stability seen across their subsets. This is not the weakness but rather is the strength of these experiments in that, they show how changing the view of the results (for example from subsets to their data sets) can affect the results of the evaluation, and hence illustrate the evaluation bias present in the effort estimation literature. This is further discussed in Chapter 4.

3.6.3 All Methods For Each Data Set

There is another way of looking at the results of these experiments and that is by looking at the trend present in each data set across different evaluation criteria. Therefore, instead of listing only the top 10 or 20 results, all methods are presented together in a graph. In order to do so, the number of losses across all subsets of a data set is summed up. This is done for each evaluation criteria. The methods are then sorted based on their number of losses for AR and the same sort order is applied to MER and MRE results. Hence, the same method is present at the same X-axis location. Figures 3.7 and 3.8 show these graphs.

These results show a clear trend in the performance of methods regardless of the evaluation criteria used. Although there are spikes present in these graphs, the general trend remains the same. It should be mentioned that, these spikes are usually small, except for three large spikes

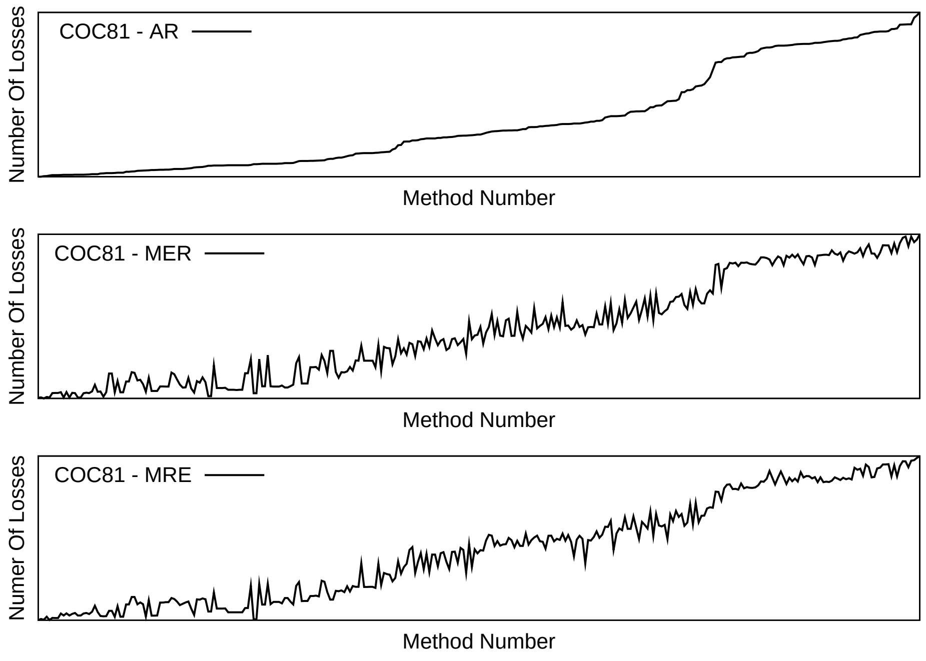

Figure 3.7: All 312 methods applied to *COC*81 and arranged using the sort order in AR.

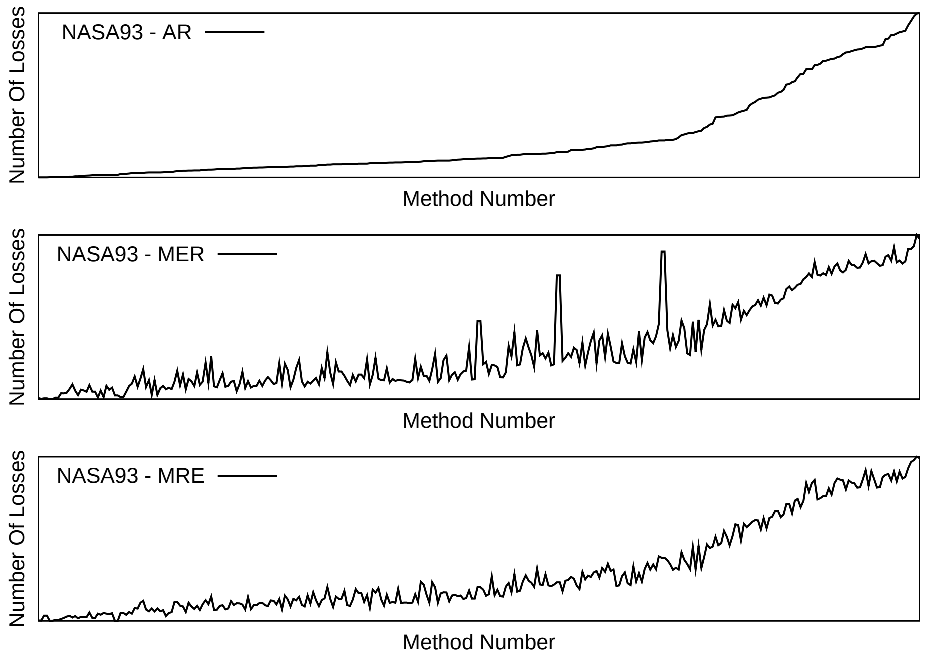

Figure 3.8: All 312 methods applied to *NASA*93 and arranged using the sort order in AR.

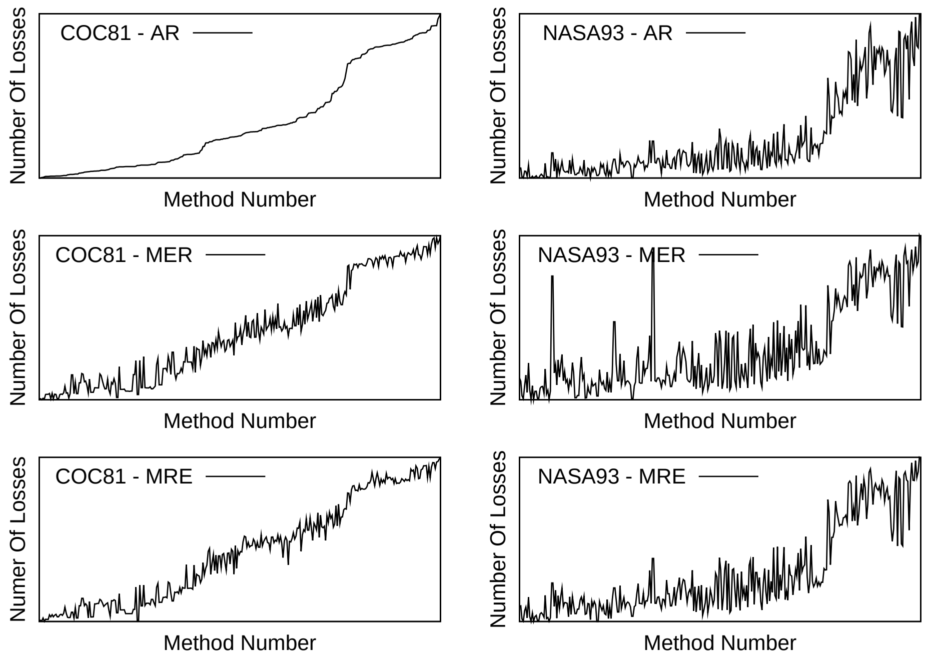

Figure 3.9: All 312 methods applied to both *COC*81 and *NASA*93 and arranged using the sort order in the top left graph.

in *NASA*93's MER results. However, these spikes are irrelevant when considering the number of methods (312) and the total number of losses possible, which is $311 * 7 = 2177$ for *COC*81 and $311 * 12 = 3732$ for *NASA*93.

Another important feature of these graphs can be seen in performance of the top performers (seen at the leftmost part of each graph). These methods are the same and usually perform at least as well as any other method. This further illustrates the similar effects of different evaluation criteria. Furthermore, it shows that the majority of the 312 methods are not needed. This clarifies the reason for showing only a small group of top methods in Tables 3.5 through 3.67 rather than showing all such methods.

As shown before, Figures 3.7 and 3.8 show a general trend across different evaluation criteria. Such a trend also exists when the same sort order is applied to the graphs of both data sets, *COC*81 and *NASA*93. This can be seen in Figure 3.9. The three graphs on the left are the same as the graphs in Figure 3.7. Therefore, they have the same spikes. However, there are more spikes, with some large ones, in the *NASA*93 graphs. This was anticipated since as stated before, a method may not perform the same way across different data sets. Nonetheless, the trend seen in either data set is similar across different evaluation criteria.

Once more, it is helpful to look at the top performing methods (seen at the leftmost part of each graph). Although they show more spikes, they still perform at least as well as all other methods. Therefore, in order to study these methods, Section 3.6.4 focuses on a smaller group of methods, namely 78 of them. Section 3.6.5 reduces the number of methods studied even further.

3.6.4 Simpler Results

Simplifying the results is helpful in many ways. Presenting too many methods makes it harder to compare the performance of these methods individually. In addition, there is a great amount of difference in the performance of certain groups of methods, as described below, such as the ones that use the superset train type or the proximal numeric estimation method. Therefore, this section aims at removing such methods and as a result, focusing on other aspects of the results of these experiments, such as finding the methods that perform worse than the others.

The results in Section 3.6.2 show that the train type can be simplified. In every case, the top performing method used the subset train type, which performed at least as well as the superset train type in the top method (Table 3.67) and otherwise much better than any method using the superset train type (Tables 3.62 through 3.65). (There might seem to be a case against removing the superset train type when looking at the tables in Section 3.6.1 since superset appears repeatedly in the results. Also, the lack of space does not allow for showing more than 10 results in each table. Therefore, in tables with only the superset train type, it seems that the superset train type performs better. However, the fact is that all those results show similar (and mostly zero) number of losses. Therefore, when looking at all, rather than top 10, methods the superset train type performs no better that the subset train type in 96.5% of the tables and only slightly better in the remaining

3.5% tables. Hence, removing the superset train type can greatly simplify the results in these tables as well. These results are available at `http://unbox.org/wisp/var/omid/extras/results/`.)

Therefore, there is a case for simplifying the results and only looking at the subset train type in the results of the remaining experiments. In addition to reducing the search space of the methods by half, it also reduces the time of the experiment by more than half. The reason is that, training on the superset takes at least as much time as, and usually more than, training on the subset (since the size of the superset is at least as large as the subset). Therefore, the time required to train each model is reduced by more than half.

There is also a case for removing the proximal numeric estimation method from further consideration in the results. As Tables 3.62 through 3.67 show, the top performing method in each data set never used proximal. This further proves the point that having some domain knowledge about the data (by using precise instead of proximal numeric estimation) helps the performance of methods. This was also seen in the case with the nearest neighbor method, which did not use the domain knowledge about the data, comparing to other methods such as local calibration.

These simplifications reduce the number of methods analyzed from 312 to a quarter of that, or 78. At this stage of reporting the results, there is no benefit in removing individual methods (such as nearest neighbor). The reason is that removing individual methods will not reduce the number of methods more than a few, which does not affect the runtime greatly. Also, the performance of such methods comparing to the top methods is worth analyzing since they are present in the literature. The next section takes a look at the best and worst performers of these 78 methods across different evaluation criteria, subsets, and an additional 4 runs.

The results of this section are generated by removing all methods with superset train type or proximal numeric estimation method and running the MWU test on the remaining 78 methods. These results are provided in Tables 3.68 through 3.73. It should be noted that the total number of losses (as well as wins and ties) possible is $77 * 7 = 539$ for $COC81$ and $77 * 12 = 924$ for $NASA93$.

Top 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
None	DynamicLocomoMean	lc	173	365	1
None	DynamicLocomoMedian	lc	189	349	1
None	None	lc	195	343	1
NewCocominMedian	DynamicLocomoMedian	lc	220	315	4
None	StaticLocomo	lc	136	398	5
NewCocominMedian	None	lc	211	322	6
NewCocominMedian	StaticLocomo	lc	164	368	7
NewCocominMedian	DynamicLocomoMean	lc	201	331	7
NewCocominNative	StaticLocomo	lc	165	366	8
OldCocomin	StaticLocomo	lc	165	366	8

Bottom 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
NewCocominMedian	NearestME	None	188	4	347
NewCocominMedian	NearestMRE	None	184	3	352
OldCocomin	NearestMRE	None	178	1	360
NewCocominNative	NearestME	None	175	1	363
NewCocominNative	NearestMRE	None	174	1	364
None	NearestMRE	None	164	7	368
None	NearestME	None	156	14	369
OldCocomin	NearestME	None	167	2	370
LocalWrapper	NearestME	None	153	3	383
LocalWrapper	NearestMRE	None	153	2	384

Table 3.68: Top 10 and bottom 10 methods for *COC*81 using AR evaluation criterion.

Top 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
None	None	cocomin1	560	363	1
OldCocomin	None	lc	560	363	1
None	None	cocomost	567	356	1
NewCocominMedian	None	lc	595	328	1
NewCocominNative	None	lc	627	295	2
None	None	cocomin2	627	295	2
OldCocomin	DynamicLocomoMedian	lc	652	270	2
NewCocominNative	DynamicLocomoMedian	lc	686	236	2
NewCocominMedian	DynamicLocomoMedian	lc	689	233	2
NewCocominMedian	StaticLocomo	lc	568	352	4

Bottom 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
OldCocomin	NearestMRE	None	435	1	488
NewCocominMedian	NearestMRE	None	406	1	517
NewCocominNative	NearestMRE	None	399	0	525
OldCocomin	NearestME	None	396	1	527
NewCocominMedian	NearestME	None	391	2	531
NewCocominNative	NearestME	None	385	0	539
None	NearestMRE	None	354	0	570
None	NearestME	None	327	0	597
LocalWrapper	NearestMRE	None	294	5	625
LocalWrapper	NearestME	None	291	5	628

Table 3.69: Top 10 and bottom 10 methods for *NASA*93 using AR evaluation criterion.

Top 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
None	DynamicLocomoMean	lc	149	386	4
None	None	lc	87	447	5
None	StaticLocomo	lc	78	454	7
None	DynamicLocomoMedian	lc	131	398	10
None	None	e	92	435	12
None	None	cocomin1	143	384	12
OldCocomin	None	lc	143	384	12
NewCocominMedian	None	lc	147	380	12
NewCocominNative	None	lc	147	380	12
None	None	cocomin2	147	380	12

Bottom 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
NewCocominMedian	NearestME	None	152	3	384
NewCocominMedian	NearestMRE	None	153	1	385
LocalWrapper	NearestME	None	149	4	386
None	NearestME	None	152	0	387
None	NearestMRE	None	146	4	389
LocalWrapper	NearestMRE	None	148	2	389
OldCocomin	NearestMRE	None	147	2	390
OldCocomin	NearestME	None	140	6	393
NewCocominNative	NearestMRE	None	145	1	393
NewCocominNative	NearestME	None	140	2	397

Table 3.70: Top 10 and bottom 10 methods for *COC*81 using MER evaluation criterion.

Top 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
None	None	cocomost	427	492	5
NewCocominMedian	None	lc	469	450	5
None	None	cocomin1	408	509	7
OldCocomin	None	lc	408	509	7
NewCocominNative	None	lc	490	424	10
None	None	cocomin2	490	424	10
None	None	lc	460	442	22
OldCocomin	DynamicLocomoMedian	lc	503	390	31
OldCocomin	StaticLocomo	lc	411	475	38
NewCocominNative	DynamicLocomoMedian	lc	553	333	38

Bottom 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
OldCocomin	NearestME	None	343	9	572
NewCocominMedian	NearestMRE	None	331	17	576
NewCocominNative	NearestMRE	None	336	12	576
NewCocominMedian	NearestME	None	332	15	577
NewCocominNative	NearestME	None	321	12	591
None	None	org	280	42	602
None	NearestMRE	None	274	1	649
LocalWrapper	NearestMRE	None	251	18	655
LocalWrapper	NearestME	None	234	18	672
None	NearestME	None	246	3	675

Table 3.71: Top 10 and bottom 10 methods for *NASA*93 using MER evaluation criterion.

Top 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
None	DynamicLocomoMean	lc	127	411	1
None	StaticLocomo	lc	83	453	3
None	None	lc	88	448	3
None	DynamicLocomoMedian	lc	122	411	6
NewCocominMedian	None	lc	146	382	11
None	None	cocomin1	146	380	13
OldCocomin	None	lc	146	380	13
NewCocominNative	None	lc	146	379	14
None	None	cocomin2	146	379	14
None	None	cocomost	146	378	15

Bottom 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
NewCocominMedian	NearestME	None	161	2	376
NewCocominMedian	NearestMRE	None	159	1	379
NewCocominNative	NearestME	None	155	1	383
OldCocomin	NearestMRE	None	155	1	383
LocalWrapper	NearestME	None	150	3	386
None	NearestMRE	None	147	5	387
OldCocomin	NearestME	None	151	1	387
None	NearestME	None	152	0	387
LocalWrapper	NearestMRE	None	149	2	388
NewCocominNative	NearestMRE	None	150	1	388

Table 3.72: Top 10 and bottom 10 methods for *COC*81 using MRE evaluation criterion.

Top 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
None	None	e	402	518	4
None	None	cocomin1	399	520	5
OldCocomin	None	lc	399	520	5
NewCocominMedian	None	lc	468	451	5
None	None	cocomost	431	487	6
None	None	lc	468	449	7
NewCocominNative	None	lc	491	424	9
None	None	cocomin2	491	424	9
OldCocomin	DynamicLocomoMedian	lc	500	409	15
OldCocomin	StaticLocomo	lc	395	512	17

Bottom 10					
Column Pruner	Row Pruner	Learner	Ties	Wins	Losses
OldCocomin	NearestME	None	306	8	610
NewCocominNative	NearestMRE	None	305	6	613
M5PWrapperNoLog	NearestME	None	298	12	614
NewCocominMedian	NearestME	None	297	11	616
M5PWrapperNoLog	NearestMRE	None	290	15	619
NewCocominNative	NearestME	None	295	5	624
None	NearestMRE	None	244	1	679
LocalWrapper	NearestMRE	None	229	9	686
LocalWrapper	NearestME	None	226	7	691
None	NearestME	None	222	1	701

Table 3.73: Top 10 and bottom 10 methods for *NASA*93 using MRE evaluation criterion.

These findings can be stated from Tables 3.68 through 3.73. For the top methods:

1. Dynamic Locomo (with mean MRE) is the row pruner of the top performing method for *COC*81 across all three evaluation criteria.

2. There is no column pruner used in the top performing method for *COC*81 across all three evaluation criteria. Nonetheless, column pruning always appeared repeatedly in the top 10 methods.

3. Hence, the same method is ranked as the top performing method for *COC*81 across all three evaluation criteria.

4. Unlike in *COC*81, in *NASA*93 and across all evaluation criteria, simple column pruning based on local calibration (such as Cocomin) was ranked in the top two methods, whether used as a pre-processor or a learner.

5. Also, unlike in *COC*81, in *NASA*93 and across all evaluation criteria, no row pruning was used in the top performing method. Nonetheless, row pruning always appeared repeatedly in the top 10 methods.

6. In both *COC*81 and *NASA*93, the top methods' learners were based on local calibration.

7. In both *COC*81 and *NASA*93, dynamic Locomo performs better than static Locomo, repeating the same finding as before.

According to these findings, row pruning using dynamic Locomo and using mean MRE as its internal evaluation method is selected for comparison in Section 3.6.5. Also, column pruning based on Cocomin (with its newer variation and using median MRE to sort) is selected for this comparison. This variation of Cocomin always performs at least as well as all other variations of Cocomin and all other column pruning methods (when looking at the number of losses in these tables). Finally, local calibration is also selected as a learner for these comparisons.

For the bottom methods:

1. In both *COC*81 and *NASA*93 and across all evaluation criteria, the nearest neighbor method constantly performed worse than any other row pruner.

2. Although Cocomin appears in both the top and bottom lists, it does so due to the presence of nearest neighbor. However, Local Wrapper never appears in the top methods and is always present in the bottom methods. Therefore, as stated before, sophisticated column pruning cannot necessarily improve the performance of methods.

3. Except for one method (in MER results for *NASA*93) all other bottom methods lack the use of a learner. (The exception actually does not use a learner, but rather uses a standard mode of COCOMO and hence, it is not really an exception to this finding.) When comparing the performance of the top and bottom methods in each table, it is clear that a learner with domain knowledge about the data (local calibration) makes a big difference in the performance of these methods.

According to these findings, the nearest neighbor method is selected as the worst performing row pruner for comparison in Section 3.6.5. Also, Local Wrapper is selected as a thorough column pruner for these comparisons.

So far, both the best and worst performing methods have been analyzed. However, there are two other methods missing from this list. These are Wrappers using LSR and M5p target learners. These methods did not appear in either top or bottom 10 methods. Therefore, there is a case for including them in the comparisons in Section 3.6.5. These methods are believed to perform better than other column pruning methods for their thorough search.

3.6.5 Top Performers For Each Data Set

This section provides the last set of results from all experiments using COSEEKMO. Thus far, all the previous sections focused on finding trends as well as best and worst performing methods

Method Number	Symbol	Column Pruner	Row Pruner	Learner
1	×	None	None	Local Calibration
2	◇	None	Dynamic Locomo	Local Calibration
3	□	Cocomin	None	Local Calibration
4	○	Cocomin	Dynamic Locomo	Local Calibration
5	△	Local Wrapper	None	Local Calibration
6	▽	LSR Wrapper	None	LSR
7	⬠	M5p Wrapper	None	M5p
8	✳	None	Nearest Neighbor	None

Table 3.74: The chosen methods from every category of methods in COSEEKMO experiments.

across different evaluation criteria and different subsets and data sets. This section focuses on another aspect of these experiments and that is the stability of results across different runs of these experiments.

The results in this section are from a total of 5 runs of COSEEKMO. These are simple runs (defined before) with the last run being the simplified version of the run that all the previous results are derived from. Each run uses a different random seed to generate the train and test sets. Except for the last run, run 5, none of the other runs were presented before. There are two reasons for doing so. One is that, all these runs showed very similar performances and hence, it was redundant to show them. The second reason is the amount of space needed for illustrating these results. However, all these results are available at `http://unbox.org/wisp/var/omid/extras/results/`.

The graphs presented in this section will include the methods discussed and selected in Section 3.6.4. These methods are provided in Table 3.74.

The graphs of the number of losses of the methods in Table 3.74 are presented in Figures 3.10 through 3.14. These graphs show the results of five different runs across three different evaluation criteria and also across two data sets and their 19 subsets. Each graph shows both *COC*81 and *NASA*93 data sets and each data sets' results are generated by finding the number of losses of each of its subsets for each specific method and graphing it. Therefore, for each method, there are 7

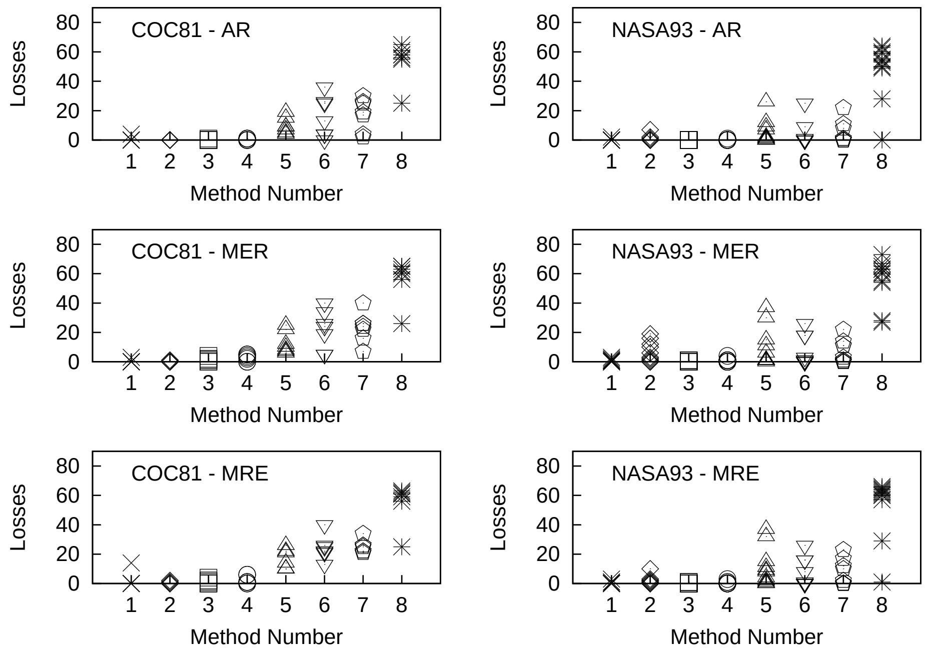

Figure 3.10: Performance of top 8 methods from Table 3.74 for run 1.

points in each of the *COC*81 graphs and 12 points for each of the *NASA*93 graphs, corresponding to the 7 and 12 subsets of each data set respectively. The lower the points are (fewer losses), the better the performance is for each method.

These graphs illustrate both the big picture and the small details of these results. The big picture is offered through categorizing 312 methods into 8 methods showing methods with best, worst, and average performances. These methods are either developed in this study as well as Baker's study [2] or represent well-known methods in the effort estimation literature. Hence, these results can offer clear comparisons of effort estimation methods. The small details help in drawing conclusions about the evaluation bias and the stability of effort estimation methods. These details are categorized into three groups of different evaluation criteria, different data set and subsets, and different runs.

There are several points that can be made about these methods:

1. At least one and usually all four methods $1,2,3,4$ perform better than all other 4 methods

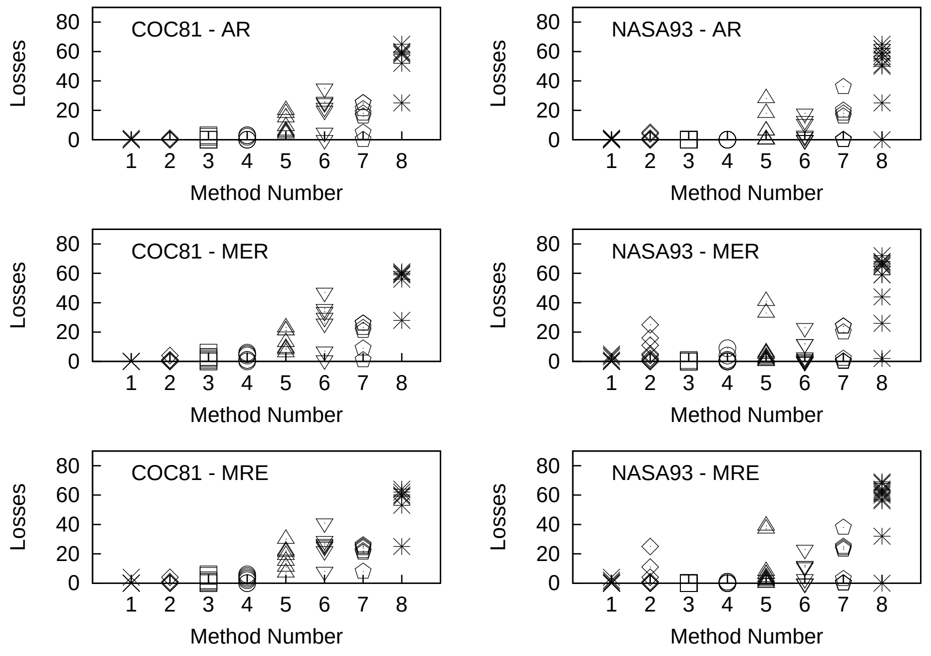

Figure 3.11: Performance of top 8 methods from Table 3.74 for run 2.

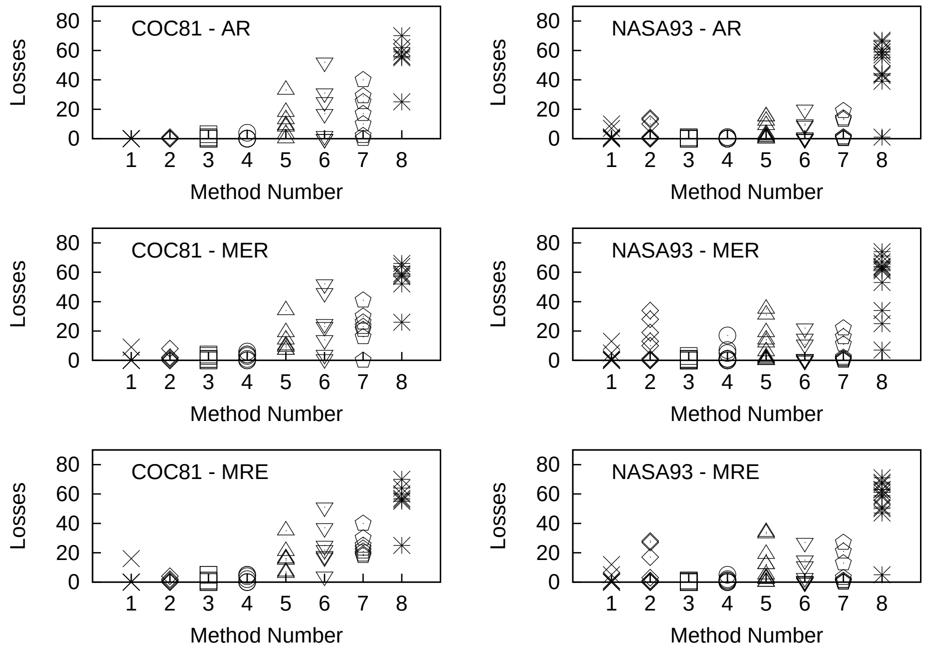

Figure 3.12: Performance of top 8 methods from Table 3.74 for run 3.

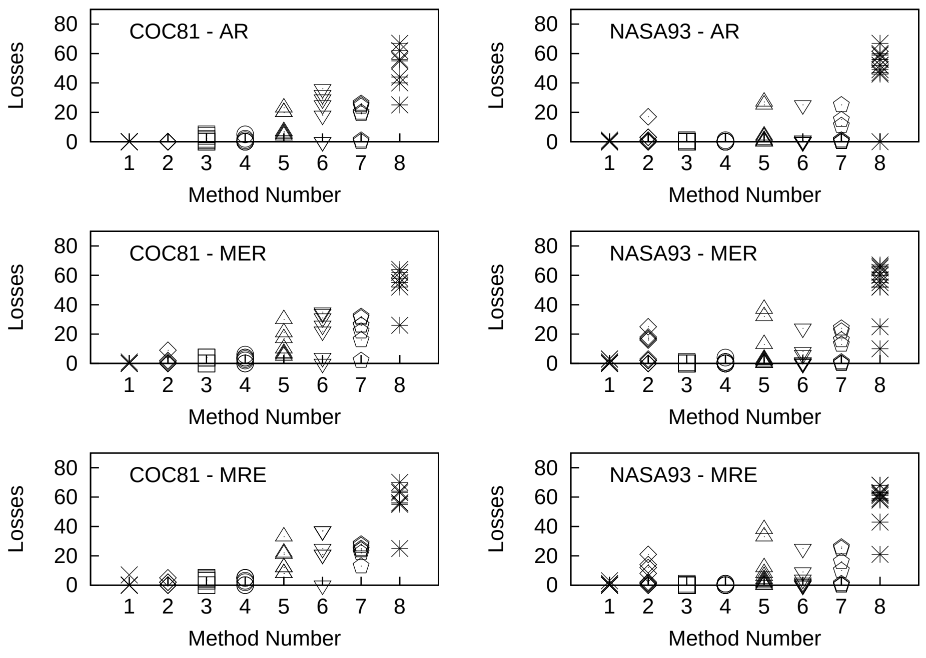

Figure 3.13: Performance of top 8 methods from Table 3.74 for run 4.

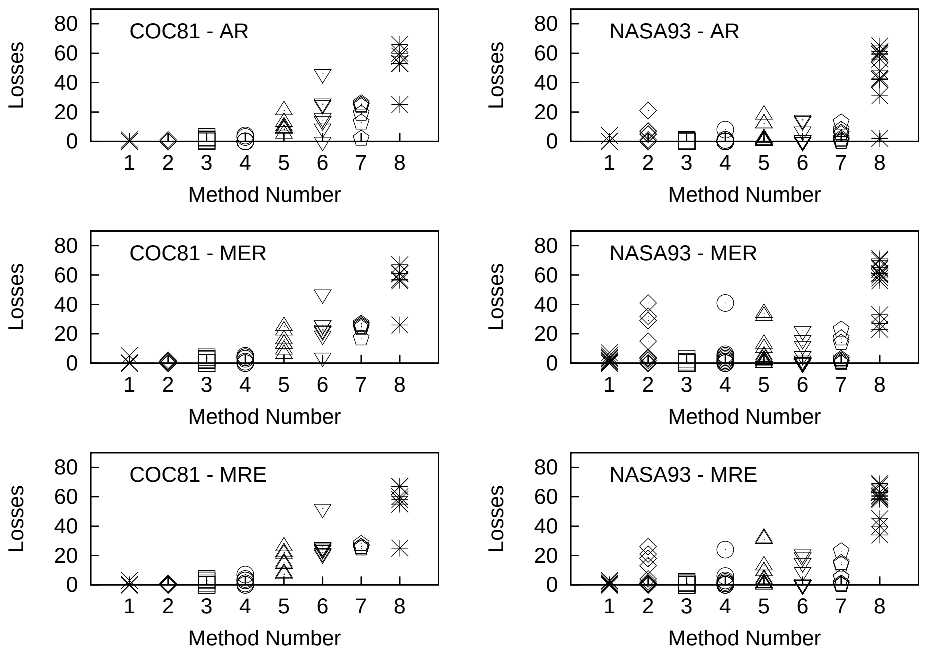

Figure 3.14: Performance of top 8 methods from Table 3.74 for run 5.

$5, 6, 7, 8$. The first four methods are based on local calibration and use zero or one simple column pruning and row pruning methods.

2. As shown in every graph, thorough column pruning based on local calibration (method 5) is always inferior to simple column pruning based on local calibration (method 3).

3. Although no conclusions can be made on the relative performance of column pruning methods with Wrappers using local calibration, LSR, or M5p (methods $5, 6, 7$), all these methods always perform worse than simple column pruners based on local calibration (method 3).

4. Method 8, which never applies the domain knowledge, always perform worse than any other method. Methods 2 and 8 are both row pruners that use a similar nearest neighbor algorithm. However, method 2 uses local calibration while method 8 does not use a learner.

5. When a row pruner (method 2) performs well, adding a column pruner (method 3) does not increase the performance of the resulting method (4). This can be seen from the $COC81$ graphs.

6. Similarly, when a column pruner (method 3) performs well, adding a row pruner (method 2) does not increase the performance of the resulting method (4). This can be seen from the $NASA93$ graphs.

7. Hence, applying one of the two pre-processors (based on local calibration) suffices.

8. Local calibration (method 1) as a learner and without any pre-processor usually performs well (and always perform better than methods not using local calibration). However, its performance is always increased when a simple row pruner (method 2) or a simple column pruner (method 3) is used as its pre-processor.

9. Model trees (method 7) usually perform similar (while they may perform slightly better or worse than) single linear regression, LSR (method 6). However, model trees always perform

worse than other single linear regression methods $(1,2,3,4)$.

Finally, there are several points that can be made about the stability of results across different evaluation criteria, data sets, and random runs. All the above findings hold across all evaluation criteria, data sets, and runs, showing a great amount of stability in the observations made when studying the biases in effort estimation. Nonetheless, as the observations become more fine-grained, biases show their nature more clearly:

1. $COC81$ responds better to row pruning (method 2) while $NASA93$ responds better to column pruning (method 3).
2. In $NASA93$, model trees (method 7) always perform worse than any simple linear regression method (including method 6). However, in $COC81$, method 7 performs similar and sometimes slightly better than method 6, although it never performs better than other simple linear regression methods.
3. Methods $1,2,3,4$ show better performances in general when evaluated using AR comparing to MER and MRE. This is also usually the case for methods $5,6,7,8$.
4. None of the runs have exactly the same performance for the any specific method across different data sets or evaluation criteria.

The interesting point is that, no matter how fine-grained these observations are made (depending on the bias), the previous findings still hold independent of the biases in the study.

Chapter 4

Conclusions

This work had two main goals. One was to study the evaluation bias in effort estimation and the other was to report the relative performance of numerous methods studied in the literature and find the best performing methods.

Studying Evaluation Bias In Effort Estimation

One of the most important biases in evaluating effort estimation methods is the type of evaluation methods used. There were several experiments with COSEEKMO comparing effort estimation methods using parametric evaluation. Those experiments illustrated how parametric evaluation methods, applied as the standard tests in the literature, were unsuitable for effort estimation experiments in general, due to their normality assumptions. As a result, this study introduced non-parametric tests as the new approach to evaluating effort estimation methods, and namely the Mann-Whitney U test. The introduction of the MWU test was beneficial in two ways:

- As a non-parametric test, it could be applied to any distribution, including the error distributions in this experiment.
- It showed a great amount of stability in comparison of different effort estimation methods,

regardless of the evaluation criteria used. This stability did not exist previously (in prior experiments as well the literature) when parametric tests were used (Section 3.3.3).

Three different evaluation criteria, AR, MER, and MRE, were used in this study. The stability of the observations made across different evaluation criteria was clear. Section 3.6 demonstrates how evaluation bias can be reduced using a non-parametric test such as the MWU test. Furthermore, applying such a test illustrated stability beyond evaluation criteria. The stability of the observations (Section 3.6.5) was also clear across different data sets and their subsets and across different random samplings of these data sets through several random runs of the experiment. (Overall, 312 combinations of methods were applied to 2 data sets with 19 subsets through 5 random runs and finally evaluated using 3 evaluation criteria.) This is contrary to what Shepperd and Kadoda [67], among other researchers [20, 36, 54], reported regarding stability.

Nonetheless, such a bias cannot be completely removed, as shown in the last findings of Section 3.6.5. The space of explored evaluation biases in this study includes some of the well-known biases. Nonetheless, there may be other evaluation biases introduced that could agree or disagree with the results of this study. Such biases can be explored in future works.

Unavoidably, due to the nature of this study, four other biases were also introduced. These biases can affect each other, and hence, affect the evaluation bias. Thus, they are worthy of a brief discussion. The first bias was in the paradigm. This study only explored the model-based methods, and among which, only the COCOMO 81 model. On the other hand, expert-based methods, which require human expertise, could not possibly be explored in a fully automatic tool such as COSEEKMO. However, a future study can be devised to compare the performance of model-based methods included in automatic tools such as COSEEKMO to those of the experts.

This study only explored the COCOMO 81 model, introducing the second bias, which is the bias in the model. However, this could not be avoided since COCOMO 81 was the only well-known software model with publicly available data. In addition, prior research using this model is extensively available, which makes it possible to measure the validity of the experiments in this study to the ones in the literature. Nonetheless, this type of bias can be explored in future works, depending on the availability of more data.

The third bias is in the selection of methods used within the model-based effort estimation methods. There are many other methods in the literature and practice that may produce different results. Although this study selected a number of methods representing several other methods, it is worthwhile to explore other methods in future works and possibly further prune the space of effort estimation methods.

The fourth and last bias involves the sampling of the data, given the fact that historical data about the projects can be maintained in a given organization. The data used in this study comes from (the only) two publicly available data sets collected by Boehm [4] and NASA (Section 3.1.1). As a result, one may argue that using only two data sets introduces the sampling bias to the observations made. However, it should be noted that these two data sets were repositories that accepted data from a wide range of projects. For example, the *NASA*93 data set comes from different teams working at geographical locations spread throughout the United States using a variety of programming languages. While some of this data is from flight systems (a particular NASA specialty), most are ground systems and share many of the properties of other terrestrial software (such as same operating systems, development languages, development practices). Much of NASA's software is written by contractors who service a wide range of clients (not just NASA). These contractors are contractually obliged (ISO-9001) to demonstrate their understanding and usage of current industrial best practices.

Nonetheless, this study explored different subsets of these data sets in order to reduce the sampling bias. Increasing the number of random runs of the experiments along with sampling at random to create different train and test sets were the measures taken along this path. Similar to other biases, this bias can be explored in future works depending on the availability of more data.

Studying Effort Estimation Methods

The comparison of effort estimation methods in this research to study evaluation bias also made it possible to explore their relative performance and report a group of methods as superior to others. Some of these methods including linear regression, model trees, and nearest neighbor (based on case-based reasoning) were studied before. Some, such as the Cocomin column pruner, were recently developed [2] and some, such as the Locomo row pruner, were developed in this study. Out of 312 combinations of methods possible in COSEEKMO, a group of 8 representatives of the above methods were selected through detailed analysis of the results in Section 3.6. This analysis made it possible to draw conclusions about the performance of these methods, which helps in simplifying the effort estimation process by limiting the scope of the search for such methods to a small list of four methods. This leads to another observation stating that there is no single best effort estimation method. Rather, there is a group of four methods that should be tried on the historical data and the best performing one can be applied to generate effort estimates for a future project.

There were several observations made throughout Section 3.6 that are organized here. These conclusions are applicable to both *COC*81 and *NASA*93 data sets and their subsets as well different evaluation criteria and different runs.

- Simple methods based on local calibration perform better than other methods.
- Applying one or both simple row pruning, using dynamic Locomo, and simple column pruning, using Cocomin, increases the performance of the local calibration learner.
- Sophisticated and thorough column pruning (using Wrappers with target learners such as local calibration, LSR, or M5p) does not increase the performance of a method and always performs worse than Cocomin.
- Row pruning without applying a learner (in the case of the nearest neighbor method that

uses case-based reasoning) produces the worst performance among all methods in this study. Hence, applying the domain knowledge is essential and therefore, similar case-based reasoning methods found in the literature are not as useful as the ones (such as Locomo) that employ local calibration. (According to [54], case-based reasoning methods belong to both sparse-data methods (requiring few or no historical data) and many-data methods (where historical data is available). Therefore, the comparison of a case-based reasoning method with other methods in this study is justified. Moreover, the same algorithm from a case-based reasoning method augmented with local calibration creates Locomo that performs well.)

- Model trees fail to increase the performance of a method in effort estimation, emphasizing the assumption of local calibration that effort is a single linear function.
- The methods that log the data (hence using linear regression) outperform the methods that do not log the data. This is one of the assumptions used in local calibration, where effort is modeled as an exponential function on the lines of code (and is linearly proportional to the product of a set of effort multipliers).
- Using proximal numeric estimation failed to increase the performance of the methods in this study. This is another assumption used in local calibration, where there is a pre-defined set of (precise) numeric values used as effort multipliers influencing the effort.
- Applying the superset train type is not useful. Methods using the superset train type usually perform no better (and generally worse) than methods using the subset train type. This could be due to the presence of more noise in the train set as it gets larger through applying the superset train type rather than the subset train type, hence reducing the performance of the trained model. As a result of this and since training using the superset takes more time, there is no reason to use the superset train type.

As a result of these observations, in order to obtain the best performance in $COC81$ and

*NASA*93 data sets, this study suggests using dynamic Locomo as a row pruner and Cocomin as a column pruner to be applied as pre-processors to local calibration. Locomo and Cocomin are fast and use simple algorithms to search for best projects and features of a data set respectively. There are a total of four combinations of these methods (no column or row pruning, only column pruning, only row pruning, and both column and row pruning) with local calibration always used as the learner. Since these are fast methods and there is no single best method among these, this study suggest applying all four combinations to the historical data and selecting the one with the best performance to generate effort estimates on future projects.

Although Locomo and Cocomin are newly developed methods, they were tested very recently and similar findings to the results of this study were reported. According to [2], when applied in the NASA's Jet Propulsion Laboratory environment, "Locomo and Cocomin when used to estimate flight projects produced a 5.4% median MRE, whereas standard LC produced a 21.97% median MRE, and SCAT provided a 29.29% median MRE." This emphasizes the validity of the findings in Section 3.6.5, where Locomo and Cocomin were found to improve the performance of local calibration. Similarly, "using Locomo to estimate ground-based projects resulted in a 15.9% median MRE, whereas standard LC and uncalibrated models such as SCAT produced 28.9% and 42.1% median MRE's, respectively [2]." Locomo is once again found to provide better performance comparing to local calibration without any row pruning.

As mentioned before, (simple) local calibration methods reported the best performance among all other methods when applied to the COCOMO 81 data sets. This is an important finding since it suggests that a data set is best explored using the models developed for it rather than more generic, and possibly sophisticated methods such as model trees, or more simplified methods such as the nearest neighbor method. Although this might seem obvious, there is a number of studies in the literature that ignore this fact. This study suggests applying or developing new and simple methods (such as Locomo and Cocomin) around the basic modeling tools (such as local calibration) developed for a software model (such as COCOMO 81) before creating more sophisticated ones that lack the domain knowledge about the data.

At the time of writing the results of this research, there is no known study that reports the stability seen in the results of the comparison of such a large number of effort estimation methods. One reason may be the fact that such a comparison is not a widespread practice due to many factors such as the lack of data, appropriate top-performing methods, and even resources. This study took more than a year and required hours of processing using computers as well as human analysis of the results. Furthermore, it required developing new methods and interacting with other researchers through this process. Therefore, many studies prefer applying one or two and usually fewer than a handful of methods and report their results based on those findings.

A second reason is the approach to comparing and evaluating the performance of effort estimation methods. As an example, even when a relatively large number of methods were compared in [46], the statistical methods applied to the results were wrong (since they used parametric tests, which this study proves them to be unsuitable). As a result, that study reported large standard deviations with $\{median, max\} = \{150\%, 649\%\}$ and identified the root of such problems to be the rare presence of large errors in the results that can mislead mean calculations. Hence, better evaluation methods (such as rejection rules in Figure 3.2) were devised to reduce such deviations. However, little progress was made to report stability in the results [46]. Other studies, such as Shepperd and Kadoda's study [67], report instability (or no stability) in their results as well.

There may be other reasons for the absence of studies reporting stability, which is considered a source of contradicting studies in the effort estimation literature. Overall, the conclusions made in studies investigating the conclusion instability problem [20, 36, 54, 67], which report doubts in the possibility of finding a solution to the instability problem (discussed in Section 2.7) were shown not to hold in this study. This study offers stability, and more importantly a framework for comparing new methods, as well as other existing methods not in COSEEKMO, with the current list of best methods reported from this work.

4.1 Future Work

There are several aspects of this research that can be further explored.

- This study only focuses on model-based methods. Expert-based methods [27] are also important to consider when comparing the performance of methods. The comparison of these two groups can only be done if experts are willing to take part in this experiment. Fortunately, the number of top-performing effort estimation methods (among the ones studied) was reduced to four in this research. Therefore, the length of such an experiment is decreased greatly, making it possible to compare expert-based methods with the four model-based methods.

- COCOMO is the only well-known and well-published software model with publicly available data. Therefore, this study only focused on COCOMO. However, there is a large number of software models that can be explored if data is available. It is suggested that COSEEKMO or a similar tool is developed for such comparisons.

- Even within COCOMO, there is no access to certain data sets. If such data sets became available, further sampling and more extensive studies can be done.

- Evaluation bias cannot possibly be explored in a single study due to the large number of evaluation methods and criteria available. Although this study explored well-known parametric tests an introduced a non-parametric test, other tests [17] and evaluation criteria can be studied in order to verify or correct the results of this study.

- The number of methods explored in this study is large. However, there are certainly other effort estimation methods to be explored. As it is seen in Figure 2.1, COSEEKMO only explored a small portion of that space. Future works can explore this even further and include methods based on neural nets or genetic algorithms. However, it is essential to study these methods systematically and offer stable conclusions or reasons for instability.

Appendix A

COSEEKMO's Manual

COSEEKMO is an open-source tool designed for performing effort estimation experiments. It consists of several effort estimation methods that are all based on COCOMO-style data sets. Although these methods are used on COCOMO-style data sets, COSEEKMO does not have such restrictions and can incorporate any method. The goal of COSEEKMO is to allow the user to compare different effort estimation methods and obtain benchmarks for new methods comparing to existing methods including the well-known methods in the literature and industry. Therefore, COSEEKMO can be expanded using new methods. This manual explains how to use COSEEKMO and how to add, remove, or modify a certain method.

A.1 Obtaining the Tool

COSEEKMO can be found at `http://unbox.org/wisp/tags/coseekmo/2.0`. All directories and their files must be downloaded for COSEEKMO to work properly. COSEEKMO is a command line tool that can be used in the Terminal of any Linux distribution such as Ubuntu (available for free at `http://www.ubuntu.com`). Although COSEEKMO was designed with Linux as the operating system in mind, Mac users can modify it to run in X11 or Terminal. Also, Windows users can use it without any modification under Cygwin (available for free at `http://cygwin.com`).

Regardless of what operating system is used, certain packages must be installed. These packages are:

- bash: the required shell for COSEEKMO,
- g++: the compiler for C++ language,
- gawk: the interpreter for the gawk language, and
- java: the interpreter for the java language. Java is used to run Weka and as of the date that this document is written, Weka requires Sun Java Runtime Environment (available for free at `http://www.java.com`) rather than the Java in combination with Linux or Gnome.

A.2 Directory Structure

COSEEKMO is divided into several parts each represented by a directory. The major files of COSEEKMO are stored in the *bin* directory. There are certain configuration files that are included in the *config* directory and are used by COSEEKMO in numeric estimations. Directories *data* and *weka* contain the data sets and the Weka data mining software (available for free at `http://www.cs.waikato.ac.nz/ml/weka`) respectively. Directory *eg* contains examples with certain methods and their combinations. Directory *evaluation* contains the evaluation tools used in COSEEKMO and specified in the *start* script.

Each method has its own directory, which includes the files necessary to run that method individually. As explained in Sections A.3 and A.4, each method has to conform to certain input and output formats. Since modifying an already existing method might not be possible, scripts are provided in the *methods* directory to modify the input and output accordingly. These scripts run the methods and are the only way that COSEEKMO uses to communicate to the methods. If a new method is added, it is strongly suggested that the existing scripts are used as templates to create new scripts for such a method. However, not all methods require scripts and some (such as column pruners) can be directly added to the COSEEKMO's main script, *generate*.

After the experiment is finished, the results are stored into *results* directory. This directory contains graphing scripts as well.

A.3 Input Format

Each method in COSEEKMO may have different input formats and accept different parameters. However, before such a method can be used in COSEEKMO, a uniform input format must be adopted. In order to facilitate the adoption of such formats, scripts can be used to convert the input provided by COSEEKMO to the input format require by the method. This eliminates the need for changes in methods' source codes and the need for recompilation. COSEEKMO provides two input files for training and testing a model as well as a string called *Prefix* that contains the structure of the output requested from the method and can be filled in with the required information provided by the method. This is further explained in Section A.4.

The train and test files are generated by COSEEKMO, enabling it to compare methods on the same input. Although each method's use for the input files may be different from the others, in a specific run all corresponding input files (whether the train or the test file) are the same for all methods. Moreover, every input file is an ARFF (Attribute-Relation File Format) file. As a part of the experiments, COSEEKMO handles the conversion of the COCOMO cost drivers from the ordinal values to numeric values used as effort multipliers. Hence, all input files contain these numeric values before they can be used by the methods.

A.4 Output Format

Since COSEEKMO is used to compare different methods, the output of all methods must conform to what COSEEKMO can use in its evaluation methods. The format of the output is passed as an input parameter to each method. The output is in CSV (Comma-Separated Value) format and each field can be completed according to the method used and the information, such as the name of the method, that is necessary to distinguish that method from other methods.

This output format can be modified if necessary as long as all methods and evaluation tools reflect this change accordingly. However, the current output format contains the necessary fields for a complete comparison of methods. These fields, in the current order are:

- a number representing the current run, which is incremented after each run in finished,
- a string representing the name of the data set (or its subset) used to generate the input files for training and testing,
- a string representing the name of the method used to create the train file,
- a string representing the name of the numeric estimation method (currently either precise or proximal),
- a string representing the name of the column pruning method used as the pre-processor (or *None* if none used),
- a string representing the name of the row pruning method used as the pre-processor (or *None* if none used),
- a number representing the number of attributes used from the train file by the method (it may be smaller than the normal value only if a column pruning method was used as a pre-processor),

- a number representing the number of instances used from the train file by the method (it may be smaller than the normal value only if a row pruning method was used as a pre-processor),
- a string representing the name of the learner used (or *None* if none used),
- a number representing the predicted value of cost for a test instance, and
- a number representing the actual value of cost for a test instance.

This ordering represents a combination of zero or one column pruning methods, zero or one row pruning methods, and zero or one learners. The method used can be any one of these three or a combination of these three.

Since column pruning can affect the number of attributes in the train file, the number of attributes in the test file is adjusted (to the number of attributes in the train file) accordingly in order to test the trained model. However, row pruning only affects the number of instances in the train file when training a model. As a result, each and every test instance can be used when testing the trained model. Hence, the number of test instances is always a constant value. Clearly, it is redundant to include these two numbers (number of attributes and number of instances of the test file) in the fields of the string above.

A.5 Adding a Method

In order to add a method, a directory can be created in the COSEEKMO tool and all the method's files can be added to that directory. As explained before, the method needs to conform to the input and output formats of COSEEKMO. Therefore, the user can add a script to the *methods* directory. This script will be responsible for communicating between COSEEKMO and the method's files stored in its directory. This way, COSEEKMO does not have to know anything about how the method works. The script handles everything the method does and only submits the results to COSEEKMO. Therefore, this script is responsible for converting the input format of COSEEKMO to the input format required by the method and also converting the output of the method to the output format that COSEEKMO requires.

There are three different types of methods that can be added to COSEEKMO.

- The column pruners are the first group. COSEEKMO calls them before calling a row pruner or a learner. They can be added to the *ColumnPrunerMethods* category in the *start* script.
- Another group is learners and row pruners that can operate on the results of a column-pruned train and test file. This group is called automatically after a column pruner has finished processing. They can be added to the *NormalMethods* category in the *start* script.
- Finally, there are methods that require no column or row pruning. They can be added to the *SpecialMethods* category in the *start* script. The benefits of this category are further explained in Section A.10.

In special cases, a method can be added directly to the main script of COSEEKMO, *generate*.

A.6 Enabling a Method

A method must be added to one of the three categories of *ColumnPrunerMethods*, *NormalMethods*, or *SpecialMethods* in the *start* script in order to be enabled.

A.7 Removing a Method

Any method can be removed from COSEEKMO by removing its directory and files as well as the scripts added to the *methods* directory. It must also be disabled by removing it from any of the three categories of *ColumnPrunerMethods*, *NormalMethods*, or *SpecialMethods* in the *start* script.

A.8 Disabling a Method

A method must be removed from any of the three categories of *ColumnPrunerMethods*, *Normal-Methods*, or *SpecialMethods* in the *start* script in order to be disabled.

A.9 Modifying a Method

Since COSEEKMO only requires certain input and output formats, modifying a method has no effect on COSEEKMO's functionality as long as the script created for the method satisfies the input and output formats. Therefore, the user can modify a method at any time.

A.10 Combination of Methods

COSEEKMO allows for combining different methods. Therefore, it is possible to create various combinations of methods in COSEEKMO. By default, COSEEKMO combines all column pruners with all the normal methods specified in the *NormalMethods* category. As briefly explained in Section A.5, a method in the *NormalMethods* category is either a learner, a row pruner, or both. Therefore, the combination of a column pruner with zero or one row pruners and zero or one learners can be done automatically by adding a method that combines zero or one row pruners with zero or one learners.

In order to make the column pruners optional, the user can add a method to the *SpecialMethods* category. This method can be a combination of zero or one row pruners with zero or one learners. Therefore, all possible combinations of zero or one column pruners, zero or one row pruners, and zero or one learners is supported in COSEEKMO.

The *SpecialMethods* category gives limitless power to the user by allowing the user to combine more than one of each type of the methods and obtain a combination that can potentially have several layers of pre-processors and learners. This allows the user to explore countless combinations, including any ordering of column pruners, row pruners, and learners.

A.11 Running the Tool

After obtaining all the files and directories of COSEEKMO (described in Section A.1), the user can run it by running the script called *start* in the main directory of COSEEKMO. This script first prepares the tool for running the methods. It then runs all the methods specified in the three categories of *ColumnPrunerMethods*, *NormalMethods*, and *SpecialMethods* and creates a log of the results. Finally, it runs the specified evaluation method and stores the results for each data set or subset in a separate file.

There are three example scripts that can be used as well. They are located in the *eg* directory. The user can run these example scripts instead of the *start* script in the main directory.

Bibliography

[1] M. Auer, C. Becker, A. Rauber, and S. Biffl. Implicit analogy-based cost estimation using textual use case similarities.

[2] Dan Baker. 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`.

[3] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. URL: `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[4] B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[5] B. Boehm. Safe and simple software cost analysis. *IEEE Software*, pages 14–17, September/October 2000. Available from `http://www.computer.org/certification/beta/Boehm_Safe.pdf`.

[6] B. Boehm and P. Papaccio. Understanding and controlling software costs. *IEEE Trans. on Software Engineering*, 14(10):1462–1477, October 1988.

[7] Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[8] Remco Bouckaert. Choosing between two learning algorithms based on calibrated tests. In *ICML'03*, 2003. Available from `http://www.cs.pdx.edu/~timm/dm/10x10way`.

[9] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. Technical report, Wadsworth International, Monterey, CA, 1984.

[10] L.C. Briand, T. Langley, and I. Wieczorek. A replicated assessment and comparison of common software cost modeling techniques. In *Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland*, pages 377–386, 2000.

[11] Zhihao Chen, Tim Menzies, Dan Port, and Barry Boehm. Feature subset selection can improve software cost estimation accuracy. *SIGSOFT Softw. Eng. Notes*, 30(4):1–6, 2005.

[12] Zhihao Chen, Tim Menzies, Dan Port, and Barry Boehm. Finding the right data for software cost modeling. *IEEE Software*, Nov 2005.

[13] S. Chulani, B. Boehm, and B. Steece. Bayesian analysis of empirical software engineering cost models. *IEEE Transaction on Software Engineerining*, 25(4), July/August 1999.

[14] P.R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.

[15] A.J.C. Cowderoy and J.O. Jenkins. Cost-estimation by analogy as a good management practice. *Software Engineering, 1988 Software Engineering 88., Second IEE/BCS Conference:*, pages 80–84, 11-15 Jul 1988.

[16] Sarah Jane Delany, Pdraig Cunningham, and Wolfgang Wilke. The limits of cbr in software project estimation.

[17] J. Demsar. Statistical comparisons of clasifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. Avaliable from `http://jmlr.csail.mit.edu/papers/v7/demsar06a.html`.

[18] Pedro Domingos and Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[19] D. Ferens and D. Christensen. Calibrating software cost models to Department of Defense Database: A review of ten studies. *Journal of Parametrics*, 18(1):55–74, November 1998.

[20] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion mmre. *IEEE Transactions on Software Engineering*, 29(11):985 – 995, November 2003.

[21] M.A. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions On Knowledge And Data Engineering*, 15(6):1437– 1447, 2003. Available from `http://www.cs.waikato.ac.nz/~mhall/HallHolmesTKDE.pdf`.

[22] R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63, 1993.

[23] S.R. Huang. Effectiveness of optimum stratified sampling and estimation in monte carlo production simulation. *Power Systems, IEEE Transactions on*, 12(2):566–572, May 1997.

[24] Certified parametric practioner tutorial. In *Proceedings of the 2006 International Conference of the International Society of Parametric Analysts, Seattle, WA*, 2006.

[25] Omid Jalali, Tim Menzies, Dan Baker, and Jairus Hihn. Column pruning beats stratification in effort estimation. *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, pages 7–7, 20-26 May 2007.

[26] R. Jensen. An improved macrolevel software development resource estimation model. In *5th ISPA Conference*, pages 88–92, April 1983.

[27] M. Jorgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70(1-2):37–60, 2004.

[28] M. Jorgensen and K. Molokken-Ostvold. Reasons for software effort estimation error: Impact of respondent error, information collection approach, and data analysis method. *IEEE Transactions on Software Engineering*, 30(12), December 2004.

[29] M. Jorgensen, D. I. K. Sjoberg, and U. Indahl. Software effort estimation by analogy and regression toward the mean. *Journal of Systems and Software*, 68(3):253–262, 2003.

[30] M. Jrgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering*, pages 33–53, January 2007.

[31] G. Kadoda, M. Cartwright, L. Chen, and M. Shepperd. Experiences using casebased reasoning to predict software project effort, 2000.

[32] Gada Kadoda, Michelle Cartwright, and Martin Shepperd. On configuring a case-based reasoning software project prediction system.

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

[34] C. Kirsopp and M. Shepperd. Case and feature subset selection in case-based software project effort prediction. In *Proc. of 22nd SGAI International Conference on Knowledge-Based Systems and Applied Artificial Intelligence, Cambridge, UK*, 2002.

[35] B. Kitchenham. A procedure for analyzing unbalanced datasets. *Software Engineering, IEEE Transactions on*, 24(4):278–301, Apr 1998.

[36] B.A. Kitchenham, L.M. Pickard, S.G. MacDonell, and M.J. Shepperd. What accuracy statistics really measure. *Software, IEE Proceedings*, 148(3):81–85, 2001.

[37] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[38] J. Li and G. Ruhe. Decision support analysis for software effort estimation by analogy. In *Proceedings, PROMISE'07 workshop on Repeatable Experiments in Software Engineering*, 2007.

[39] K. Lum, J. Hihn, and T. Menzies. Sudies in software cost model behavior: Do we really understand cost model performance? In *ISPA Conference Proceedings*, 2006. Available from `http://menzies.us/pdf/06ispa.pdf`.

[40] K. Lum, J. Powell, and J. Hihn. Validation of spacecraft software cost estimation models for flight and ground systems. In *ISPA Conference Proceedings, Software Modeling Track*, May 2002.

[41] Carolyn Mair and Martin J. Shepperd. Looking at comparisons of regression and analogy-based software project cost prediction. In *Software Engineering Research and Practice*, pages 113–118, 2006.

[42] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.*, 18(1):50–60, 1947. Available on-line at `http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&handle=euclid.aoms/1177730491`.

[43] T. Menies, K. Lum, and J. Hihn. The deviance problem in effort estimation. In *PROMISE, 2006*, 2006. Available from `http://menzies.us/06deviations.pdf`.

[44] T. Menzies and J. Hihn. Evidence-based cost estimation for better quality software. *IEEE Software*, July/August 2006. Available on-line at `http://menzies.us/pdf/06costs.pdf`.

[45] T. Menzies, D. Port, Z. Chen, J. Hihn, and S. Stukes. Validation methods for calibrating software effort models. In *Proceedings, ICSE*, 2005. Available from `http://menzies.us/pdf/04coconut.pdf`.

[46] Tim Menzies, Zhihao Chen, Jairus Hihn, and Karen Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, November 2006. Available from `http://menzies.us/pdf/06coseekmo.pdf`.

[47] Tim Menzies, Zhihao Chen, Dan Port, and Jairus Hihn. Simple software cost estimation: Safe or unsafe? In *Proceedings, PROMISE workshop, ICSE 2005*, 2005. Available from `http://menzies.us/pdf/05safewhen.pdf`.

[48] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, January 2007. Available from `http://menzies.us/pdf/06learnPredict.pdf`.

[49] A. Miller. *Subset Selection in Regression (second edition).* Chapman & Hall, 2002.

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

[51] K. Molokken and M. Jorgensen. A review of surveys on software effort estimation. In *ISESE'03*, 2003.

[52] Tridas Mukhopadhyay, Steven S. Vicinanza, and Michael J. Prietula. Examining the feasibility of a case-based reasoning model for software effort estimation. *MIS Q.*, 16(2):155–171, 1992.

[53] Ingunn Myrtveit and Erik Stensrud. Do arbitrary function approximators make sense as software prediction models? In *STEP '04: Proceedings of the 12 International Workshop on Software Technology and Engineering Practice (STEP'04)*, pages 3–9, Washington, DC, USA, 2004. IEEE Computer Society.

[54] Ingunn Myrtveit, Erik Stensrud, and Martin Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineerining*, 31(5):380–391, May 2005.

[55] R. Park. The central equations of the price software cost model. In *4th COCOMO Users Group Meeting*, November 1988.

[56] L. Pickard, B. Kitchenham, and S. Linkman. An investigation of analysis techniques for software datasets. *Software Metrics Symposium, 1999. Proceedings. Sixth International*, pages 130–142, 1999.

[57] Rahul Premraj, Martin Shepperd, and Michelle Cartwright. Meta-data to guide retrieval in cbr for software cost prediction. In *8th UK Workshop on Case-based Reasoning*, pages 26–37, December 2003.

[58] Rahul Premraj, Martin Shepperd, and Michelle Cartwright. Data enrichment in case-based reasoning for software cost prediction. In *PREP 2004*, April 2004.

[59] L. Putnam and W. Myers. *Measures for Excellence*. Yourdon Press Computing Series, 1992.

[60] J. R. Quinlan. Learning with Continuous Classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992. Available from `http://citeseer.nj.nec.com/quinlan92learning.html`.

[61] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992. ISBN: 1558602380.

[62] B. Boehm S. Chulani, B. Clark and B. Steece. Calibration approach and results of the cocomo ii post-architecture model. In *Proceceedings ISPA,98*, 1998.

[63] Thomas Saaty. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. McGraw-Hill International Book Co, Singapore, 1980.

[64] M. Shepperd. Software project economics: A roadmap. In *International Conference on Software Engineering 2007: Future of Software Engineering*, 2007.

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

[66] M. Shepperd and G. Kadoda. Using simulation to evaluate prediction techniques [for software]. *Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International*, pages 349–359, 2001.

[67] M. Shepperd and Gada F. Kadoda. Comparing software prediction techniques using simulation. *IEEE Trans. Software Eng*, 27(11):1014–1022, 2001.

[68] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12), November 1997. Available from `http://www.utdallas.edu/~rbanker/SE_XII.pdf`.

[69] Martin Shepperd, Chris Schofield, and Barbara Kitchenham. Effort estimation using analogy. In *ICSE '96: Proceedings of the 18th international conference on Software engineering*, pages 170–178, Washington, DC, USA, 1996. IEEE Computer Society.

[70] Spareref.com. Nasa to shut down checkout & launch control system, August 26, 2002. `http://www.spaceref.com/news/viewnews.html?id=475`.

[71] Peter Sprent. *Applied Nonparametric Statistical Methods*. Chapman & Hall, London, 1993.

[72] The Standish Group Report: Chaos, 1995. Available from `http://www4.in.tum.de/lehre/vorlesungen/vse/WS2004/1995_Standish_Chaos.pdf`.

[73] R. Strutzke. *Estimating Software-Intensive Systems: Products, Projects and Processes*. Addison Wesley, 2005.

[74] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, dec 1945.

[75] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[76] Ian H. Witten and Eibe Frank. *Data mining. 2nd edition*. Morgan Kaufmann, Los Altos, US, 2005.

[77] Yisehac Yohannes. *Classification and Regression Trees, Cart: a User Manual for Identifying Indicators of Vulnerability to Famine and Chronic Food Insecurity*. Intl Food Policy Research Inst, 1999.