A Principled Methodology: A Dozen Principles of Software Effort Estimation

by

Ekrem Kocaguneli

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

Doctor of Philosophy
in
Computer Science

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2012

Keywords: Software Effort Estimation , Machine Learning, Regression Trees, Regression, Neural Nets, Analogy, *k*-NN, Ensemble, Kernel

Abstract

A Principled Methodology: A Dozen Principles of Software Effort Estimation

by

Ekrem Kocaguneli
Doctor of Philosophy in Computer Science

West Virginia University

Tim Menzies, Ph.D., Chair

Software effort estimation (SEE) is the activity of estimating the total effort required to complete a software project. Correctly estimating the effort required for a software project is of vital importance for the competitiveness of the organizations. Both under- and over-estimation leads to undesirable consequences for the organizations. Under-estimation may result in overruns in budget and schedule, which in return may cause the cancellation of projects; thereby, wasting the entire effort spent until that point. Over-estimation may cause promising projects not to be funded; hence, harming the organizational competitiveness.

Due to the significant role of SEE for software organizations, there is a considerable research effort invested in SEE. Thanks to the accumulation of decades of prior research, today we are able to identify the core issues and search for the right principles to tackle pressing questions. For example, regardless of decades of work, we still lack concrete answers to important questions such as: "What is the best SEE method?" The introduced estimation methods make use of local data, however not all the companies have their own data, so: "How can we handle the lack of local data?" Common SEE methods take size attributes for granted, yet size attributes are costly and the practitioners place very little trust in them. Hence, we ask: "How can we avoid the use of size attributes?" Collection of data, particularly dependent variable information (i.e. effort values) is costly: "How can find an essential subset of the SEE data sets?" Finally, studies make use of sampling methods to justify a new method's performance on SEE data sets. Yet, trade-off among different variants is ignored: "How should we choose sampling methods for SEE experiments?"

This thesis is a rigorous investigation towards identification and tackling of the pressing issues in SEE. Our findings rely on extensive experimentation performed with a large corpus of estimation techniques on a large set of public and proprietary data sets. We summarize our findings and industrial experience in the form of 12 principles:

1) Know your domain
2) Let the Experts Talk
3) Suspect your data
4) Data Collection is Cyclic
5) Use a Ranking Stability Indicator
6) Assemble Superior Methods
7) Weighting Analogies is Over-elaboration
8) Use Easy-path Design
9) Use Relevancy Filtering
10) Use Outlier Pruning
11) Combine Outlier and Synonym Pruning
12) Be Aware of Sampling Method Trade-off

Acknowledgements

Firstly, I would like to thank my PhD committee chair and my supervisor Dr. Tim Menzies. It was a blessing to be one of his PhD students and be able to conduct research with him. I would not be able to succeed without his patience, support and contagious love for research. He has a tremendous effect on my research, on my life and on me as a person. I am forever in hist debt for his unmatched passion as a supervisor, a researcher and a friend.

I would like to thank Dr. Arun Ross, Dr. Bojan Cukic, Dr. Katerina Goseva and Dr. Mark Culp for their invaluable input to my research and for being in my committee. Their guidance helped me immensely to shape and improve my research as well as its presentation in this document.

I am thankful to my lab mates at Modeling and Intelligence Laboratory of West Virginia University. It was a fantastic experience to be among these passionate and hardworking friends. They were always a source of friendship, support and ideas for me.

I would like to acknowledge the help of everyone working in the Lane Department of Computer Science and Electrical Engineering for making it one of the nicest departments to be at. I would also like to present my sincere thanks to West Virginia University and its staff for providing all the wonderful facilities and services, which helped me on a day to day basis during my research as well as during my daily life in the beautiful city of Morgantown.

I am grateful to my previous supervisor Dr. Ayse Basar Bener, who was very instrumental for me to begin my PhD and who has a considerable influence on me as a person and as a researcher. I am also in debt to all my collaborators for different parts of my research. I would like to present special thanks to Dr. Jacky W. Keung, Dr. Tom Zimmermann, Dr. Christian Bird, Dr. Nachiappan Nagappan, Dr. Jairus Hihn, Dr. Emilia Mendes, Dr. David Cok and Dr. Ray Madachy.

Finally, I would like to thank my mother Zeliha, my father Ali and my brother Mehmet Ali, who supported me all throughout my life. Whenever I needed their support, they were always there for me with their unconditional love. Last but not least, I thank Gizem Erdogan, who was there with me during all my graduate studies.

Contents

List of Figures

List of Tables

Notation

The notation and the symbols used throughout this document are as follows:

SE	:	Software Engineering
SEE	:	Software Effort Estimation
ML	:	Machine Learning
ABE	:	Analogy-based Estimation
$IRWM$	:	Inverse-ranked weighted mean
$norm$	:	Normalization
log	:	Take natural logarithm
SFS	:	Sequential Forward Selection
$SWReg$	:	Stepwise Regression
$CART$	:	Classification and Regression Trees
$NNet$	:	Neural Net
$SLReg$	:	Simple Linear Regression
PCR	:	Principal Component Regression
$PLSR$	:	Partial Least Squares Regression
$TMPA$	:	Theoretical Maximum Prediction Accuracy
$B\&V$	:	Bias and Variance
FP	:	Function Points

Chapter 1

Introduction

Software effort estimation is an important field of empirical software engineering and it can be defined as the activity of correctly estimating the effort required to complete a software project. In the Introduction, we will introduce the problem of software effort estimation and explain why it is critical for the success of contemporary software development. We will continue with defining how the accumulation of more than 3 decades of software effort estimation research points out the open questions of the field. Our research aims at investigating the open issues of software effort estimation. The following paragraphs of the Introduction will briefly introduce our research as well as the principles that emerged as a result of this research. Furthermore, after each principle and the research that led to this principle, we will briefly mention the related contributions of that research.

Correctly estimating the effort required to develop software is of vital importance. Over or under-estimation of software development effort can lead to undesirable results:

- Under-estimation results in schedule and budget overruns, which may cause project cancellation.
- Over-estimation hinders the acceptance of promising ideas, thus threatening organizational competitiveness.

Since effort estimation plays such an important role for software companies, a considerable effort has been invested in the issue. According to Jorgensen and Shepperd the largest research topic in software effort estimation (hereafter, SEE) is the introduction of new methods and their comparison and evaluation to existing ones [3]. In their comprehensive review of the SEE literature, Jorgensen and Shepperd report that more than 60% of the peer reviewed SEE studies deal with this topic [3].

Regardless of the huge amounts of research effort invested in introduction of new methods, we still search for answers to a set of questions. For example, “What is the best SEE method?” The introduced estimation methods make use of local data, however not all the companies have their own data, so: “How can we handle the lack of local data?” SEE methods take size attributes for granted, yet size attributes are costly and the practitioners place very little trust in them. Hence, we ask: “How can we avoid the use of size attributes?” Collection of data, particularly dependent variable information (i.e. effort values) is costly: “How can find an essential subset of the SEE data sets?” Finally, studies make use of sampling methods to justify new methods performance on SEE data sets. However, we do not evaluate the trade-off between different sampling methods, so we ask: “How should we choose sampling methods for SEE experiments?”

Shepperd et al. mentions the problem of not having concrete answers to the afore-mentioned issues and identifies 3 major areas of focus [4] and points to likely culprits of that problem:

- Model accuracy
- The dataset
- Sampling methods

To that list, we make the addition of “data collection”, i.e. how should one collect and look at the data?

Our work for the past 3 years has been a rigorous research towards the key issues and related questions of SEE. This document is the outcome of such an investigation.

We summarize our ideas and findings in the form of 12 principles. The principles regarding data collection in an industrial setting (Principles 1 to 4) are our ideas and experiences that we would like to share in a less technical manner compared to the other principles. Principles 5 to 12 (inclusive) are the summary of our rigorous experimental research, which led to the principle recommended at the end of each chapter.

We have no claim that the principles presented in this document are absolute, yet they provide clearer answers for a large corpus of SEE data sets and evaluation criteria. To the best of our knowledge, the experiments reported in each chapter make use of most of the available public data sets for SEE research and all the widely used error measures. Therefore, the principles, recommendations and contributions reported as a result of the rigorous experimentation reported in this document concern a large portion of SEE research domain. The following is a list of the identified principles as a result of the research presented in this thesis and the related contributions.

The first 4 principles concern the data collection activities. Unfortunately, it is rarely the case that the data that an SEE practitioner wants to use is readily available [5]. In the case where the SEE practitioner requires to mine his/her own data from the repositories of an organization, the mining of the data becomes an interplay of domain knowledge, interaction with experts as well as the update of the past collection and analysis with the emergence of new information. The related principles of the first 4 chapters share our ideas and experiences regarding the data collection activities. The principles that summarize our discussion in each related chapter are:

Principle #1: Know your domain

Principle #2: Let the experts talk

Principle #3: Suspect your data

Principle #4: Data Collection is Cyclic

In the related chapters, each of these principles will be further discussed. In summary, the contributions of the data collection related discussions can be summarized as follows:

Contributions:

- Sharing of hands-on experience with industrial data collection
- Identification of possible pitfalls and likely solutions

It is difficult to find an SEE method that performs well across different data sets and different error measures, i.e. methods are bound to change their ranking under changing conditions [4]. We evaluate a high number of methods (90 methods as introduced in §2.4) induced on a high number of data sets (20 public data sets of Figure 2.1) and evaluated w.r.t. 7 error measures (as described in §2.3). This large scale evaluation confirmed the prior cautions that changing conditions (i.e. change of data set and/or change of error measure) results in rank changes of methods. However, we were also able to observe that a small subset of the methods were more successful and more stable compared to all the rest. The following principle summarizes this research:

$\boxed{\text{Principle \#5: Use a Ranking Stability Indicator}}$

The contributions of the research behind this principle:

- A method to identify successful SEE methods using their rank changes.
- An evaluation method of the diversity of the SEE data sets.
- To show that use of a ranking stability indicator can help to identify successful methods both for aggregate and specific cases.

As a result of the prior principle, we have a list of the small subset of methods that are more stable and more successful than the rest. Such a knowledge shall be used to advance the performance of solo-methods (a solo-method is a combination of a pre-processor and a learner). Reading from the literature of machine learning (ML), we see that knowledge of successful methods can be used to form ensembles, which are expected to improve the performance of the individual members of the ensemble [6–8]. Unlike prior SEE studies on ensemble methods that ignore to find the superior solo-methods [9–11], our research (which targets superior solo-methods) reports statistically significant performance improvements through the use of ensembles. Therefore, we recommend the following principle:

$\boxed{\text{Principle \#6: Assemble Superior Methods}}$

The contributions of the research on ensemble methods are:

- A novel scheme for ensembling the best solo-methods.
- Stable multi-methods that outperform *all* solo-methods.

Another promising direction of research in terms of successful SEE research is the investigation of analogy-based estimation (ABE). ABE research is heavily invested in by prominent research groups of SEE [4, 12–15]. There are multiple beneficial factors of ABE methods that account for this interest, such as [15]: 1) No model-calibration to local data is required, i.e. ABE can work with limited local data; 2) ABE can handle outliers, which are common in SEE datasets; 3) ABE can choose similar projects with 1 or more attributes, so it can work even if all the attributes of a new project are not yet defined; 4) It is not based on a parametric model, so it is useful if the domain is difficult to model; and, 5) ABE process is similar to human reasoning, hence it is easy to explain to a customers. The problem with the ABE methods is that the space of design options related to different ABE methods is enormous. For example, in [16], we show that ABE0-like methods can exceed *17,000* different design options. Hence, it is important to discover this this space. A recent interest in ABE methods is to elaborate on the analogies (i.e. the training instances that are used for estimation) [1, 17]. An extensive experimentation with kernel-weighting of the analogies showed that such an elaboration has virtually no benefit. Hence, we state the following principle:

Principle #7: Weighting Analogies is Over-elaboration

The contributions of the research that led to the 7^{th} principle are:

- Investigation of an unexplored and promising ABE option of kernel-weighting.
- An extensive experimentation of 2090 ABE scenarios, which reduces the ABE variants space to be explored.

Unlike over-elaborating on the details of ABE methods, we observed in multiple scenarios that the design principles associated with ABE methods prove very beneficial [16, 18]. Such a design principle, which also names the next principle is the so-called *"easy-path design"*. Easy-path design asks the user to identify the fundamental assumptions of a particular ABE variant and then get rid of the training instances that violate these assumptions. Once the assumption-violating training instances are removed, the ABE variant's prediction accuracy has been observed to increase significantly. The 8^{th} principle recommends the easy-path design as a principle:

Principle #8: Use Easy-path Design

The contributions of the research that used easy-path design principle to design an ABE variant called dynamic-ABE (D-ABE) are:

- An ABE design principle that can be applied to different ABE methods.
- Discovering the performance space between *static-k* based ABE methods and TMPA.

Besides the performance of estimation methods, another fundamental issue that is to be dealt with in SEE is the availability of the local data (local data can be defined as the data of an organization) [2, 19]. Most of the fundamental SEE models are built on the available local data, e.g. COCOMO [20]. However, the availability of local data is not always the case. In such cases, the local data of other organizations or different time intervals (so called cross data) can be of help. There is a significant amount of research effort invested in enabling the transfer of cross data to local domains [2, 19, 21–23]. There is not an agreed-upon solution to the transfer of cross data, neither is there an agreement on the success of the cross data use. However, recently it was seen that relevancy filtering applied on the cross data improves the performance of using cross data to the extent of using local data [22]. Our extensive experiments on public as well as proprietary data sets using a relevancy-filtering method called TEAK showed that we can enable the transfer of cross data without comprising from the performance. Hence, for cross data usage, we recommend:

Principle #9: Use Relevancy Filtering

The contributions of the research behind this principle are:

- Utilization of a novel method for time interval transfer.
- Evaluation of the proposed transfer learning method on recent proprietary as well as public data sets; hence, providing evidence for practitioners as well as benchmark availability for further research.
- To evaluate the previous cross-company research in SE from a transfer learning perspective.

Another issue that is relevant with the data collection and model building in SEE is the sizing attributes of SEE data sets. Widely used SEE models like COCOMO and function points (FP) are all built on the premise of the availability of size attributes such as lines of code or logical transactions, respectively. However, -as we will discuss further in Chapter 10- the fact that size attributes are at the heart of SEE methods create a considerable opposition among the practitioners and disrupts the adoption of SEE methods. Hence, it is critically important for SEE to promote methods that can work without size attributes. A popularity-based ABE method called pop1NN identifies the popular instances (training instances that are closest neighbors to other instances) and prunes the rest. Such an SEE method can compensate the lack of size attributes in SEE data sets. Therefore, we recommend the following:

Principle #10: Use Outlier Pruning

The contributions of this research are as follows:

- Promotion of SEE methods that can compensate the lack of the software size features.
- A method called pop1NN that shows that size features are not a “must”.

Using *pop1NN* in order to compensate for the size attributes in SEE and using *easy-path design* principle have common attributes. Both of these research share the common property that they get rid of the noise within the training data and they hint that there is an easier structure to SEE data. Therefore, we question the essential content of the SEE data sets, in other words what is the least amount of training data in terms of instances and features that can provide the same performance as a method using the entire training data? That question bears a fundamental importance for SEE, because the majority of the SEE research is invested in discovery of new methods [3] and often times the justification of the complexity of the proposed methods is overlooked. By improving the pop1NN to identify popular instances as well as popular features (so called synonyms), we propose an algorithm called QUICK. QUICK prunes away unpopular instances and popular features to reduce SEE data sets to an essential content. Our experimental results show that QUICK can attain performance values as good as standard ABE methods and classification-and-regression-trees, by using only a fragment of the data set. Therefore, we recommend the following principle to find the essential content of SEE data sets and suggest that complex methods that provide marginal performance improvements should be justified.

Principle #11: Combine Outlier and Synonym Pruning

The contributions of the research searching for the essential content of SEE data sets are:

- An unsupervised method to find the essential content of SEE data sets and reduce the data needs.
- Promoting research to elaborate on the data, not on the algorithm.

All the SEE methods promoting a new method use a sampling method (SM) in order to separate the data set into training and test sets. However, different sampling methods pose different bias-variance values, different run times and different degrees of ease-of-replication. The trade off between these factors is overlooked and this issue poses a threat to the comparison of different studies as well as replication of the work. The issue of the effects of different SMs has only been discussed by Kitchenham et al. previously [21], yet there is no study evaluating different SMs through extensive experimentation. Our recommendation for the SEE community is to use a specific SM, leave-one-out cross validation. The more general recommendation as a result of our research is:

Principle #12: Be Aware of Sampling Method Trade-off

The contributions of the experimental investigation of the trade-off between different SMs are:

- The first systematic investigation of *B&V* trade-off in SEE domain
- An extensive experimentation with 20 public datasets and 90 algorithms
- Showing that *B&V* trade-off and run times of SMs are not the main concerns for SEE
- Recommendation based on experimental concerns:
  - For reproducibility, we prefer LOO since this avoids non-deterministic selection of train and test sets.

1.1 Statement of This Thesis

Accumulation of the 30+ years of software effort estimation work enables us to identify the core issues as well as the overlooked ones, e.g. what is the best effort estimation method, how should we advance analogy-based estimation methods, how can we handle local data requirements, how can we choose the appropriate sampling method for software effort estimation? This thesis identifies and answers such core issues.

1.2 Structure of This Thesis

This thesis is structured as follows: Various chapters share common material such as the properties of the data sets and error measures used in this research, pre-processors and learners used in experimentation as well as background notes on SEE. Such common material is discussed in Chapter 2. Common validity issues are discussed in Chapter 3. This is followed by the recommended principles, where each chapter presents the research as well as the principle that is recommended at the end of this chapter. The industrial data collection related notes, which merely share our experiences from projects with industry are presented in Chapter 4. Starting from Chapter 5 we present our research leading to the principle that is recommended at the end of each chapter. Each of these chapters (Chapter 5 to Chapter 12) start with a paragraph explaining the research that will be presented throughout the chapter. The chapters finish with a conclusion, which names the principle and lists the contributions of the research presented in that chapter. Finally, we present the future directions to our research as well as the conclusions of our research Chapter 13.

1.3 Publications from This Thesis

The following is a list of publications written as a result of the research summarized in this thesis.

JOURNAL PAPERS

IEEE Transactions on Software Engineering:

- E. Kocaguneli, T. Menzies, J. Keung, *"On the Value of Ensemble Effort Estimation"*, IEEE Transactions on Software Engineering, 2011.
- E. Kocaguneli, T. Menzies, A. Bener, J. Keung, *"Exploiting the Essential Assumptions of Analogy-based Effort Estimation"*, IEEE Transactions on Software Engineering, 2011.

Empirical Software Engineering Journal:

- E. Kocaguneli, T. Menzies, J. Keung, *"Kernel Methods for Software Effort Estimation"*, Empirical Software Engineering Journal, 2011.

Automated Software Engineering Journal:

- J. Keung, E. Kocaguneli, T. Menzies, *"A Ranking Stability Indicator for Selecting the Best Effort Estimator in Software Cost Estimation"*, Journal of Automated Software Engineering, 2012.

Currently Under Review:

- E. Kocaguneli, T. Menzies, J. Keung, *"Active Learning for Effort Estimation"*, third round review at IEEE Transactions on Software Engineering.
- E. Kocaguneli, T. Menzies, E. Mendes, *"Transfer Learning in Effort Estimation"*, submitted to IEEE Transactions on Software Engineering.
- E. Kocaguneli, T. Menzies, *"Software Effort Models Should be Assessed Via Leave-One-Out Validation"*, under second round review at Journal of Systems and Software.
- E. Kocaguneli, T. Menzies, E. Mendes, *"Towards Theoretical Maximum Prediction Accuracy Using D-ABE"*, submitted to IEEE Transactions on Software Engineering.

CONFERENCE PAPERS

- E. Kocaguneli, T. Menzies, J. Hihn, Byeong Ho Kang, *"Size Doesn't Matter? On the Value of Software Size Features for Effort Estimation"*, Predictive Models in Software Engineering (PROMISE) 2012.
- E. Kocaguneli, T. Menzies, *"How to Find Relevant Data for Effort Estimation"*, International Symposium on Empirical Software Engineering and Measurement (ESEM) 2011
- E. Kocaguneli, G. Gay, Y. Yang, T. Menzies, *"When to Use Data from Other Projects for Effort Estimation"*, International Conference on Automated Software Engineering (ASE) 2010, *Short-paper.*

Chapter 2

Background

Throughout the thesis, there are common materials that are shared by different chapters. For example, the background information regarding SEE is common to almost all of the experiments; hence, it will be presented here. Another common material is the performance measures used to evaluate different SEE methods. Also, some data sets are common to different chapters. Instead of repeating such information, we opted for creating a background part, under which different chapters present related material. In this chapter we also present background information regarding a large set of learners and pre-processors that are commonly used in SEE research. Due to various advantages of ABE methods on SEE data sets, our discussion of ABE methods are more in-depth, compared to other methods.

2.1 Software Effort Estimation (SEE)

Software effort estimation (SEE) can be defined as the process/activity of estimating the total effort necessary to complete a software project [24]. According to the extensive systematic review conducted by Jorgensen and Shepperd, developing new models is the biggest research topic in SEE since 1980s [3]. Therefore, there are many SEE models that have been proposed over the years and a taxonomy is necessary to classify such a large corpus. Myrtveit et al. defines taxonomy as an explanation of a concept by highlighting the similarities and differences between that particular concept and the related ones [14].

There exists a number of different taxonomies proposed in the literature [14, 25]. Briand et al. report that there is no agreement on the best taxonomy and define all proposed taxonomies to be subjective and to have flaws [25]. For example Menzies et al. divide SEE methods into two groups: Model-based and expert-based [26]. According to this taxonomy model-based methods use some algorithm(s) to summarize old data and to make predictions regarding the new data. On the other hand, expert-based methods make use of human expertise, which is possibly supported by process guidelines and/or checklists.

Myrtveit et al. use a different taxonomy, where they propose a dataset dependent differentiation between methods [14]. According to that taxonomy the methods are divided into two:

- Sparse-data methods that require few or no historical data: e.g. expert-estimation [27], automated case-based reasoning [28].
- Many-data approaches where certain amount of historical data is a must: e.g. functions and arbitrary function approximations (such as classification and regression trees).

Shepperd et al. propose a 3-class taxonomy [28]: 1) expert-based estimation, 2) algorithmic models and 3) analogy. According to this taxonomy expert based models target the consensus of human experts through some process like Delphi [29]. Jorgensen et al. define expert-based methods as a human-intensive process of negotiating the estimate of a new project and arriving at a consensus [27]. There are formal methods proposed for expert-based estimation like Delphi [29]. However, Shepperd et al. notes in another study that it is mostly the case that companies follow an informal process for expert-based estimation [4]. Algorithmic models include the adaptation of a formula to local circumstances or local data. Prominent examples to these methods are the COCOMO method [20] and function points [30]. Analogy based methods include finding past projects that are similar to the current project that is to be estimated and then adapting the effort values of these past projects.

Regardless of the taxonomy used to group SEE methods under different classes, the ultimate goal of all the methods are to generate realistic estimates. In [31] Jorgensen defines some guidelines for generating realistic software effort estimates. An important finding in Jorgensen's study (which parallels the findings behind *"Principle #6: Assemble Superior Methods"* in Chapter 6) is that *combining estimations* coming from different sources (e.g. from experts and instance-based learners) captures a broader range of information related to the estimation problem.

2.2 Datasets

Figure 2.1 lists 20 real world software development project datasets, which are publicly available. As shown in Figure 2.1, public data sets are composed of a variety of data sets coming from different parts of the world and collected via different methodologies. The public data sets used for various experiments reported in this document are: The standard **COCOMO** data sets (cocomo*, nasa*), which are collected with the COCOMO approach [20]. The **desharnais** data set, which contains software projects from Canada. It is collected with function points approach. **SDR**, which contains data from projects of various software companies in Turkey. SDR is collected by Softlab, the Bogazici University Software Engineering Research Laboratory [32]. **albrecht** data set consists of projects completed in IBM in the 1970's and details are given in [33]. **finnish** data set originally contains 40 projects from different companies and data were collected by a single person. The two projects with missing values are omitted here, hence we use 38 instances. More details can be found in [34]. **kemerer** is a relatively small dataset with 15 instances, whose details can be found in [35]. **maxwell** data set comes from finance domain and is composed of Finnish banking software projects. Details of this dataset are given in [36]. **miyazaki** data set contains projects developed in COBOL. For details see [37]. **telecom** contains projects which are enhancements to a U.K. telecommunication product and details are provided in [38]. **china** dataset includes various software projects from multiple companies developed in China.

For the Chapter 9, we required to identify cross and within data source. We define *cross-within sources* as the subset(s) of effort data sets that are formed through division of one feature: Instances having the same value for that feature form a subset. Such features are plausible candidates for generating a *cross* source experiment, i.e. the features should be likely to change from one source to other. Accordingly, we explored public as well as proprietary data sets manually. After manually inspecting 20 public datasets of Figure 2.1, six were found to be suitable for *cross-within* experimentation. Those six data sets support the 21 *cross-within divisions* shown in Figure 2.2. The selected division criteria include:

- project type: embedded, organic and semidetached (*cocomo81*),
- center: geographical development center (*nasa93*),

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

Figure 2.1: The 1198 projects coming from 20 public data sets. Indentation in column one denotes a dataset that is a subset of another dataset.

- language type: programming language used for development (*desharnais*),
- application type: on-line service program, production control program etc. (*finnish* and *maxwell*),
- hardware: PC, mainframe, networked etc. (*kemerer* and maxwell),
- source: whether in-house or outsourced (maxwell).

We will use the following nomenclatures: If a subset name is followed by a set of numbers, they correspond to values of the feature used to form the subset. If a name has multiple numbers at the end (e.g. finnishAppType2345) then all instances with these values are combined in a single subset.

The cross-company experiments on public datasets are also repeated for proprietary data sets from Tukutuku data base. Tukutuku data base is used for transfer learning between domains [12]. Tukutuku brings together data sets coming from a high number of companies. The version used in

Dataset	**Criterion**	**Subsets**	**Subsets Size**
cocomo81	project type	cocomo81e	28
		cocomo81o	24
		cocomo81s	11
nasa93	development center	nasa93_center_1	12
		nasa93_center_2	37
		nasa93_center_5	39
desharnais	language type	desharnaisL1	46
		desharnaisL2	25
		desharnaisL3	10
finnish	application type	finnishAppType1	17
		finnishAppType2345	18
kemerer	hardware	kemererHardware1	7
		kemererHardware23456	8
maxwell	application type	maxwellAppType1	10
		maxwellAppType2	29
		maxwellAppType3	18
maxwell	hardware	maxwellHardware2	37
		maxwellHardware3	16
		maxwellHardware5	7
maxwell	source	maxwellSource1	8
		maxwellSource2	54

Figure 2.2: 6 datasets are selected from 20 candidates. Then selected datasets are divided into subsets according to a criterion that can define a *cross-within division*. The datasets, subset sizes as well as the selection criteria are provided here.

this research is composed of a total of 195 projects developed by a total of 51 companies. However, not all the companies in the data set are useful for cross data analysis. We eliminated all the companies with less than 5 projects, which yielded 125 projects from 8 companies. The abbreviations used for the 8 companies that were selected and their corresponding number of projects are given in Figure 2.3.

The Tukutuku data base is an active project, which is maintained by Emilia Mendes. The data set includes information collected from completed Web projects [39]. These projects come from a total of 10 different countries around the world [40]. The majority of the projects in Tukutuku data base are new development (65%), whereas the remaining ones are enhancement projects. Tukutuku data base is characterized by a total of 19 independent variables and a dependent variable. The dependent variable is the total effort in person hours used to develop an application.

We were also interested to see how the transfer learning experiments between domains would work for between time intervals. So as to see the transfer learning performance and selection tendency between time intervals, we used 2 data sets: Cocomo81 and Nasa93. Each of these data sets are divided into 2 subsets of different time periods. The subsets of Cocomo81 are: coc-60-75 and coc-76-rest, where *coc* stands for Cocomo81, *60-75* stands for projects developed from 1960 to 1975 and *76-rest* stands for projects developed from 1976 onwards. It is possible to have different divisions of the data depending on different time frames. Our selection selects these particular divisions of time periods so that both subsets span a certain amount of time (e.g. more than a decade) and yet have at least 20 instances. The subsets of nasa93 are: nasa-70-79 and nasa-80-rest. The naming convention is similar to that of Cocomo81 subsets: *nasa* stands for Nasa93 dataset, *70-*

Company	**# of Projects**
tuku1	14
tuku2	20
tuku3	15
tuku4	6
tuku5	13
tuku6	8
tuku7	31
tuku8	18

Figure 2.3: The abbreviations that will be used for selected companies and the corresponding number of projects.

79 stands for projects developed in the time period of 1970 to 1979 and *80-rest* stands for projects developed from 1980 onwards. The details of these projects are provided in Figure 2.4.

Dataset	**Features**	**Size**	**Description**
coc-60-75	17	20	Nasa projects between 1960 and 1975
coc-76-rest	17	43	Nasa projects from 1976 onwards
nasa-70-79	17	39	Nasa projects between1970 and 1979
nasa-80-rest	17	54	Nasa projects from 1976 onwards
		Total: 156	

Figure 2.4: Data sets for transfer learning over time period.

2.3 Error Measures

There are multiple performance measures (a.k.a. error measures) used in SEE. In this research, we use a total of 8 error measures. Performance measures aim to measure the success of a prediction. For example, the absolute residual (AR) is the absolute difference between the predicted and the actual:

$$AR_i = |x_i - \hat{x}_i| \tag{2.1}$$

(where $x_i, \hat{x}_i$ are the actual and predicted value for test instance i). We use a summary of AR through taking the mean of AR, which is known as Mean AR (MAR).

The Magnitude of Relative Error measure a.k.a. MRE is a very widely used performance measure for selecting the best effort predictor from a number of competing software prediction models [38, 41]. MRE measures the error ratio between the actual effort and the predicted effort and is expressed by the following equation:

$$MRE_i = \frac{\mid x_i - \hat{x}_i \mid}{x_i} = \frac{AR_i}{x_i} \tag{2.2}$$

A related measure is MER (Magnitude of Error Relative to the estimate [41]):

$$MER_i = \frac{\mid x_i - \hat{x}_i \mid}{\hat{x}_i} = \frac{AR_i}{\hat{x}_i} \tag{2.3}$$

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

$$MMRE = mean(allMRE_i) \tag{2.4}$$

$$MdMRE = median(allMRE_i) \tag{2.5}$$

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

$$PRED(25) = \frac{100}{N}\sum_{i=1}^{N}\begin{cases} 1 \text{ if } MRE_i \leq \frac{25}{100} \\ 0 \text{ otherwise} \end{cases} \tag{2.6}$$

For example, PRED(25)=50% implies that half of the estimates fall within 25% of the actual values [38].

Other performance measures used in this thesis are Mean Balanced Relative Error (MBRE) and the Mean Inverted Balanced Relative Error (MIBRE), both suggested by Foss et al. [41]:

$$MBRE_i = \frac{|\hat{x_i} - x_i|}{min(\hat{x_i}, x_i)} \tag{2.7}$$

$$MIBRE_i = \frac{|\hat{x_i} - x_i|}{max(\hat{x_i}, x_i)} \tag{2.8}$$

The above mentioned performance measures are selected due to their wide use in the SEE research. However, none of these error measures are devoid of problems. For instance, MRE-based error measures have been criticized due to their asymmetry [41]. This criticism applies to MMRE, MdMRE and Pred(25). Rather than going into the debate of which error measure is better than the others, we evaluated our results subject to a total of 8 error measures. A recent study by Shepperd et al. provides an excellent discussion of the error measures [42]. In this study Shepperd et al. propose a new unbiased error measure called Standardized Accuracy (SA), which is based on the mean absolute error (MAE). SA's equation is as follows:

$$SA = 1 - \frac{MAE_{P_i}}{\overline{MAE_{P_0}}} \tag{2.9}$$

MAE_{P_i} is defined to be the MAE of the estimation method P_i. $\overline{MAE_{P_0}}$ is the mean of a large number of (in our case 1000) random guessing. In the random guessing procedure a training instance is randomly chosen with equal probability from the training set (with replacement) and its effort value is used as the estimate of the test instance. SA gives us an idea of how good an estimation method is in comparison to random guessing. Since the term MAE_{P_i} is in the nominator, the higher the SA values, the better an estimation method.

Interpreting these error measures without any statistical test may be misleading. A recent discussion about this issue can be found in [43]. To evaluate our results subject to a statistical test, we make use of so called *win-tie-loss* statistics. Win-tie-loss statistics employ either a Wilcoxon Signed Rank or a Mann-Whitney Rank Sum non-parametric statistical hypothesis test with 95% confidence. These tests are more robust than the Student's *t*-test as they compares the signs or sums of ranks, unlike Student's *t*-test, which may introduce spurious findings as a result of outliers in the given datasets. Non-parametric tests are also useful, if it is not clear that the underlying distributions are Gaussian [44].

We store the performance of every method w.r.t. each error measure used over each dataset. This enables us to collect *win-tie-loss* statistics using the algorithm of Figure 2.5. In Figure 2.5, we first check if two distributions i, j are statistically different according to the appropriate statistical test (Wilcoxon or Mann-Whitney at 95% confidence); if they are not, then we increment tie_i and tie_j. If the distributions are statistically different, we update win_i, win_j and $loss_i, loss_j$ after comparing their error measures.

```
if NonParametricTest($E_i$, $E_j$, 95) says they are the same then
    $tie_i = tie_i + 1$;
    $tie_j = tie_j + 1$;
else
    if better($E_i$, $E_j$) then
        $win_i = win_i + 1$
        $loss_j = loss_j + 1$
    else
        $win_j = win_j + 1$
        $loss_i = loss_i + 1$
    end if
end if
```

Figure 2.5: Comparing methods (*i*,*j*).

The $better$ function in the if statement of Figure 2.5 varies according to the performance criteria. For some error measures such as MMRE and MdMRE, "better" means lower values, i.e. lower means and medians respectively. For PRED(25), "better" means higher PRED(25) values.

2.4 Pre-Processors and Learners

In different chapters, we will be referring to the results of Chapter 5, where 90 solo-methods are compared. A solo-method is a combination of a pre-processor with a learner. The 90 solo-methods come from the combination of 10 pre-processors and 9 learners. In this chapter, we define and explain the different pre-processors and learners.

2.5 Pre-Processors

The 10 pre-processors used for investigation in this thesis are:

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

None is just the simplest *option* of avoiding a pre-processor, i.e. all data values are unadjusted.

With the **norm** (max-min) preprocessor, numeric values are normalized to a 0-1 interval using Equation 2.10. Normalization means that no variable has a greater influence than any other.

$$normalizedValue = \frac{(actualValue - min(allValues))}{(max(allValues) - min(allValues))} \tag{2.10}$$

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

Principal component analysis [45], or **PCA**, is a *feature synthesis* preprocessor that converts a number of possibly correlated variables into a smaller number of uncorrelated variables called components. The first component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.

Some of the preprocessors aim at finding a subset of all features according to certain criteria such as **SFS** (sequential forward selection) and **SWR** (stepwise regression). **SFS** adds features into an initially empty set until no improvement is possible with the addition of another feature.

Whenever the selected feature set is enlarged, some oracle is called to assess the value of that set of features. In this study, we used the MATLAB, *objective* function (which reports the the mean-squared-error of a simple linear regression on the training set). One caution to be made here is that exhaustive search algorithms over all features can be very time consuming (2^n combinations in an n-feature dataset), therefore SFS was designed to work only in forward direction (no backtracking).

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

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

- **width3bin:** This procedure clumps the data features into 3 bins, depending on equal width of all bins see Equation 2.11.

$$binWidth = ceiling\left(\frac{max(allValues) - min(allValues)}{n}\right) \tag{2.11}$$

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

2.6 Predictors (Learners)

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

- x2 *instance-based* learners: **ABE0-1NN, ABE0-5NN**;
- x2 *iterative dichotomizers*: **CART(yes),CART(no)**;

- x1 *neural net*: **NNet**;
- x4 *regression methods*: **LReg, PCR, PLSR, SWR**.

Instance-based learning can be used for analogy-based estimation (ABE). ABE0 is our name for a very basic type of ABE that we derived from various ABE studies [1,46,47]. In **ABE0-*k*NN**, features are firstly normalized to 0-1 interval, then the distance between test and train instances is measured according to Euclidean distance function, *k* nearest neighbors are chosen from training set and finally for finding estimated value (a.k.a adaptation procedure) the median of *k* nearest neighbors is calculated. We explore two different *k*NN:

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

Iterative Dichotomizers seek the best attribute value $splitter$ that most simplifies the data that fall into the different splits. Each such splitter becomes a root of a tree. Sub-trees are generated by calling iterative dichotomization recursively on each of the splits. The CART iterative dichotomizer [48] is defined for continuous target concepts and its $splitters$ strive to reduce the GINI index of the data that falls into each split. In this study, we use two variants:

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

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

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

2.7 Analogy-Based Effort Estimation (ABE) and ABE0

Analogy-based estimation (ABE) is a specific case of case-based reasoning (CBR) and it appears to be a low cost alternative to more complex learners [49]. Furthermore, in 2 chapters of this thesis (Chapter 8 and Chapter 9) ABE methods play a fundamental role, whereas in other chapters ABE methods are used in comparison to different estimation methods. Therefore, in this chapter we provide the background information of CBR and specifically ABE, then we continue with the definition of a baseline ABE method, which is called as "'ABE0'.

CBR is an active research field [4,12,13,50], which can be defined as the activity of estimating the new cases by using similar past cases [4,51,52]. CBR uses a so called knowledge-base (a.k.a. case-base) of past instances. Knowledge-base is used to *"retrieve"* one or more similar past cases (as well as their solutions) to the current problem at hand [53]. In our case the knowledge-base is the training set. Then CBR *"reuses"* the retrieved cases to propose a solution for the current problem [54]. Once the solution is *"revised"*, the final estimation is performed. A typical CBR system would *"retain"* this new experience of problem solving in its knowledge-base [52]. These steps constitute a CBR-cycle, which is given in Figure 2.6.

In summary, CBR is a 4-step procedure:

1. *RETRIEVE* the most similar cases from the case base, i.e. choose the most similar past projects to the new project, which is to be estimated.

2. *REUSE* the past case information to propose a solution for the new case, i.e. get the effort values of past projects to estimate the new project.

3. *REVISE* the solution of the previous stage, i.e. make further corrections on the proposed effort estimate.

4. *RETAIN* the solution for the new case, i.e. store the information of the new project in the dataset.

2.7.1 Analogy-based Effort Estimation

Analogy based estimation (ABE) or estimation by analogy (EBA) is a form of case based reasoning (CBR). In their 2005 study Myrtveit et. al. follow the taxonomy of grouping SEE methods under two main classes: sparse-data and many-data methods [14]. With respect to this taxonomy CBR may belong to both classes depending on how past experience is used. For example if CBR is used to reason from an already selected case, then it is grouped under sparse data methods. On the other hand, if CBR is used to identify and adapt the past case(s), then it is a many-data method. ABE is an example of CBR being used as a many-data method.

ABE has been used extensively in SEE [4, 12, 13, 16, 50]. It has been shown that ABE methods are able to attain comparable or even better performance values than more complex algorithms [13, 15, 16]. Aside from the performance point of view, there are many other advantages of ABE over the other more complicated methods [15]:

- No model-calibration to local data is required, i.e. ABE can work with limited local data.

- ABE can handle outliers, which are common in SEE datasets

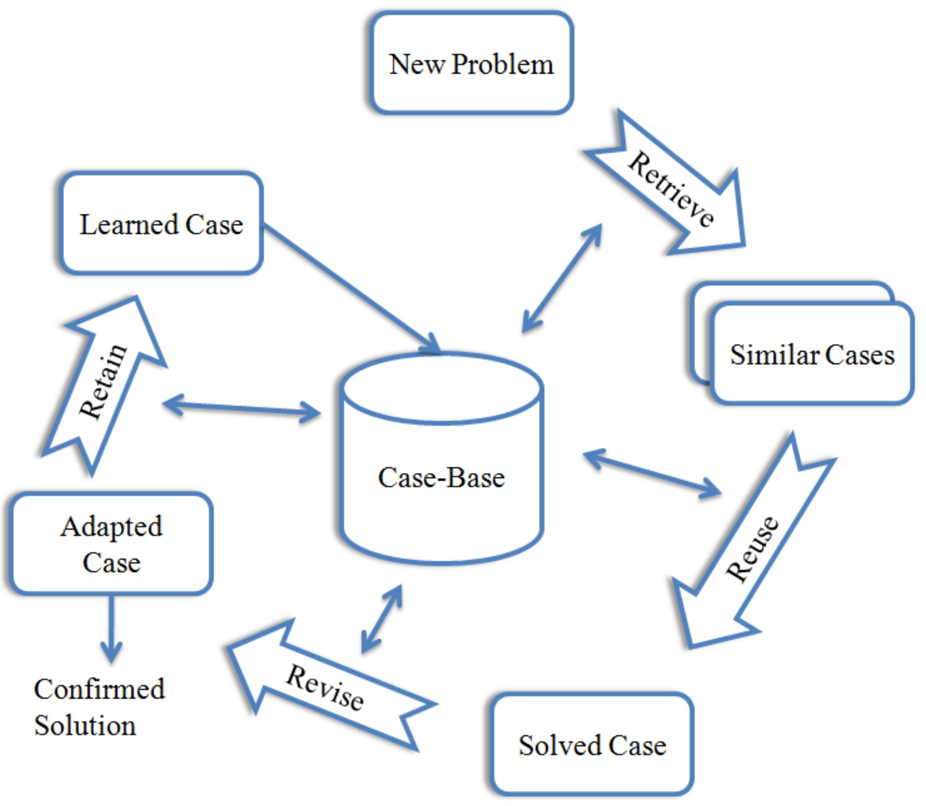


Figure 2.6: The CBR life-cycle. At the arrival of a new problem, similar cases are *"retrieved"* from the *"case-base"*. Retrieved cases are *"reused"* and *"revised"* to reach a confirmed solution. Depending on the correctness of the estimate, solution may be *"retained"* in the case-base as a learned-case for future use.

- ABE can choose similar projects with 1 or more attributes, so it can work even if all the attributes of a new project are not yet defined
- It is not based on a parametric model, so it is useful if the domain is difficult to model
- Finally, ABE process is similar to human reasoning, hence it easy to explain to a customer.

2.7.2 Baseline ABE: ABE0

ABE has a number of design options corresponding to different stages of CBR. Some of the design options are:

- **The case subset selection**, i.e. noise removal: e.g. 1) remove nothing [38], 2) use outlier removal [13], 3) use prototype methods to generate a representative set [55].

- **Feature selection methods**: e.g. 1) using genetic algorithms to learn useful features [46], 2) exhaustive methods like wrapper [56] and 3) various filter methods.
- **Feature weighting methods**: e.g. some weighting methods use initial discretization methods such as 1) equal-frequency discretization, 2) equal width discretization, 3) entropy [57], 4) PKID [58] and 5) use features as is, i.e. do nothing at all.
- **The similarity function**, i.e. different measures used to decide the similarity between projects, e.g.: 1) weighted and 2) unweighted Euclidean distance, 3) triangular distribution-based distance function by Frank et al. [59], 4) Minkowski distance [60] and 5) the mean value ranking [15].
- **Adaptation mechanisms**, i.e. how to use the effort values of the closest projects, e.g.: 1) take mean, 2) take median, 3) use a second learner for adaptation [26, 61] and 4) report the weighted mean [1]
- **The methods to select analogies**, i.e. how many analogies to use and how to choose them: 1) Using a static number of analogies (e.g. Li et al. [46] recommends using $1 \leq k \leq 5$ analogies), 2) dynamic number of analogies for a particular test instance [16, 61].

Note that just considering the above options, we have:

3 subset selectors
$\times$ *3 feature selectors*
$\times$ *5 feature weighting schemes*
$\times$ *5 similarity functions*
$\times$ *4 adaptation mechanisms*
= 900 ABE variants.

Following Kadoda&Shepperd [47], Mendes et al. [1], and Li et al. [46] we can define a baseline ABE called ABE0, which executes the following steps:

- Input a database of past projects
- For each test instance, retrieve *k* similar projects (analogies).
  - For choosing *k* analogies use a similarity measure.

  - Before calculating similarity, scale independent features to equalize their influence on the similarity measure.
  - Use a feature weighting scheme to reduce the effect of less informative features.

- Adapt the effort values of the *k* nearest analogies to come up with the effort estimate.

ABE0 uses the Euclidean distance as a similarity measure, whose formula is given in Equation 2.12, where w_i corresponds to feature weights applied on independent features. ABE0 framework does not favor any features over the others, therefore each feature has equal importance in ABE0, i.e. $w_i = 1$. For adaptation ABE0 takes the mean of retrieved *k* projects.

$$Distance = \sqrt{\sum_{i=1}^{n} w_i(x_i - y_i)^2} \tag{2.12}$$

Chapter 3

Validity Issues

The results presented in a research are not devoid of validity threats due to multiple reasons and our research presented in this document is not an exception. This chapter briefly discusses the validity issues associated with the use of particular methods and data sets in this thesis.

Construct validity (i.e. face validity) assures that we are measuring what we actually intended to measure [62]. Previous studies have concerned themselves with the construct validity of different performance measures for effort estimation (e.g. [41]). While, in theory, these performance measures have an impact on the rankings of effort estimation predictors, we have found that other factors dominate. For example, Figure 5.4 showed that some of the datasets have a major impact on what could be concluded after studying a particular estimator on these datasets. Note that the data sets used for the experimentation of this thesis have been extensively used in SEE community by other research groups as well. Based on our experience with working these data sets as well as based on our reading of the related literature using these data sets, we have no reason to believe that there are fundamental construct validity issues associated with the collection of these data sets.

External validity is the ability to generalize results outside the specifications of a certain study [63]. To ensure external validity, different research ideas presented in this document have utilized a large number of projects coming from virtually all the available public SEE data sets as well as additional proprietary ones. Our data sets are diverse, measured in terms of their sources, their domains and the time they were developed in. We use datasets composed of software development projects from different organizations around the world to generalize our results [32]. Our reading of the literature is that we use more data, from more sources, than numerous other papers.

For example, Table 4 of [64] list the total number of projects used by a sample of other studies. The median value of that sample is 186; i.e. only a fraction of the 1200+ projects used in the experimentation of this thesis.

Internal validity asks to what extent the cause-effect relationship between dependent and independent variables holds [45]. The ideal case to observe that relationship would be to learn a theory on the available data and apply the *learned* theory on completely new and unseen data. So as to simulate the case of new and unseen data, sampling methods are employed.

It is clear that this research has not explored absolutely all the range of effort estimation methods. Future work may be helpful to repeat some of the research ideas in this document using the top performing solo-methods found here and possibly more. Nevertheless, given the extent of our experimentation, this thesis offers much more support compared to other SEE studies.

Another validity issue to mention (regarding the use of solo-methods) is that none of learners have been exhausted via fine-tuning. Therefore, future work is required to exhaust all the parameters of every learner to use their best version. However, exhaustive fine-tuning of a learner through some heuristic may be as comprehensive as a paper on its own right, e.g. tabu search heuristic proposed by Corazza and Mendes et al. [40].

Chapter 4

Ground Zero: When Do I Have the Perfect Data?

At the end of an industry presentation, one of the attendees directed me the question: "When do you know you have the perfect data?". This is a very valid question for any estimation task that requires the use of past data. Unfortunately there is no formula or silver-bullet answer. This is mainly due to the fact that reaching the right data is an interplay of various different factors such as domain knowledge; people generating, using and collecting the data; particular characteristics of the data. Hence, the principles we will use are along these lines. This brief chapter and the principles recommended are less technical and less experimental compared to other chapters. However, any researcher or practitioner is likely to face situations in an actual data collection setting, where the recommended principles may come reasonable and helpful. This chapter is a result of rewording the initial question into the following form: "When do I have the perfect data?" and it aims to share our experiences.

4.1 Principle #1, Know your domain

Each domain or product has a way of reflecting on the collected data data. Sometimes what seems to be a recurring anomaly has a reasonable explanation. However, this knowledge is usually hidden to an outsider data analyst. Fayyad et al. highlight the importance of domain knowledge even more [65]. They note that *"Use of domain knowledge is important in all steps"* in knowledge discovery.

It is vitally important to get an idea of the processes behind the observed data. However, such knowledge is hidden in the particular domain of an organization and it takes time and effort to learn and discover this knowledge. For example, if one is to analyze the commit information of a particular company, it is best to spend some time understanding the generation and progression of their code in that corporation's particular branching structure. In one of the projects that we were involved in, there would be users associated with tens of commits per a business day. This information may indeed be true, e.g. some developers prefer to commit small changes just to be on the safe side. Yet, it needed to be supplemented with domain knowledge. Further investigation via domain experts revealed that these users were responsible for the progression of the code in the branching structure and their commits were not related to code generation. Therefore, asking the right questions such as which users or user types touch the code in the branching structure, how is this captured in the data bases were important in that particular project to reveal important indicators.

Another example can be the collection of effort data from a company's project management software. Mere use of the numbers stored in the data bases may be misleading. For example, in an industry project "planned time" and "actual time spent" by developers had 100% match, which is in fact probably the ideal case from a project manager's perspective. However, a talk with the domain experts revealed that the 100% match is merely a way developers used the tool, i.e. regardless of the actual time spent for particular task, the time information that would be filled out by the developers was only a re-type of the planned number. Hence, this information needed to be updated and/or augmented with further investigation.

One further example to the requirement of domain knowledge in data collection emerged during the productivity analysis for a commercial organization. An analysis of the percentage of edits committed by each developer revealed that a handful of developers in different projects were responsible for more than 80% of the entire commits. In other words, the numbers were telling that the projects were developed by less than 5 people, although there were close to 100 developers involved in each project. This information obviously needed to be further clarified. Therefore, we went and talked with the project managers and department heads to make sure that the observed pattern in data was a reflection of the truth. They were also surprised to see such a pattern and suggested that this may indeed be the case, yet they recommended that we talked to the actual developers (finding the right person to talk to for the domain knowledge). When we talked with the developers who were responsible for more than 80% of the commits, the fact revealed itself: The commit structure in different teams was made in such a way that most of the new developers would transfer their edits to the more experienced developers for a sanity check. Experienced developers would then make sure that there was not anything fundamentally wrong with the edits of the less experienced developers. Then the experienced developers would perform the commit. Hence, the observed pattern was indeed the reflection of the processes employed among the developers, yet the numbers could have been interpreted much differently without the accompanying domain knowledge.

It is possible to increase the number of examples, where suspicious information has perfectly normal explanation through particular domain properties. However, the main idea is to get the domain knowledge right. On the other hand, getting the right domain knowledge may not be as easy as it seems. It requires one to ask the right questions to the right people. If possible, getting to talk with multiple domain experts and cross-validating the learned domain knowledge proves to be beneficial.

4.2 Principle #2, Let the Experts Talk

Very initial results have a higher chance of being blown by the domain experts, this is only natural. Because, the very initial results are based on the data with limited domain knowledge and with limited user input. Remembering from the war stories mentioned in the previous section, the initial analysis is likely to be the analyst's interpretation of the numbers and it may be very different from the reality. Therefore, initial results or the initial analysis may be off the target. Yet, the initial results presented in the early meetings of the project present fundamental opportunities.

The success of the very initial results are better to be measured by the amount of discussions they stimulate among the experts or attendees of that meeting. If it creates a discussion and you are bombarded with questions and suggestions regarding how to improve your analysis, then the analysis is on a good track and this is the time to listen to experts regarding what false assumptions were made during data collection, what other considerations should be included and so on. Furthermore, it may be a good time to know the right people to ask domain specific questions. Such discussions and involvement also provide opportunity for the next projects. Multiple times the next project to be funded was decided during the ongoing project, upon the discussions that the domain experts built on top of the presented results, i.e. *let the experts talk*.

A caution regarding this principle is to make sure that all the experienced domain experts are aware of the initial results. Because, experienced experts have domain knowledge and their time is valuable. Therefore, they may choose to skip the initial meetings due to other priorities. On the other hand, it may create fundamental flaws in the data collection not to have their input and not to provide them the opportunity to talk. For example, during a project we needed to find out the amount of code added to different work packages in a large software platform. The problem was that work packages are associated with different work groups and different projects that were all working on the same platform. The solution that emerged after the discussion with the experienced domain experts that were attendees of the initial meetings was that although there is the possibility that different teams may be committing to the same work package simultaneously, in reality that was never the case. In other words it was safe to assume that the commits performed on a work package during the course of a project were performed by a single team. An analysis of some sample projects confirmed this assumption, so we built further analysis on this assumption. However, after about 50% of the project was complete, an experienced domain expert, who was responsible for a specific set of projects and who was unable to attend the prior meetings opposed this assumption. He was able to identify a number of projects from his own domain that violated this assumption, which resulted in finding ways to clean the data, then recollecting it and updating the entire analysis until that point. Hence, although the experts may skip the initial meetings due to more urgent priorities, it is best to organize one-to-one meetings with such experts to provide them an opportunity to talk and make sure that they are on board with the current results and assumptions.

4.3 Principle #3, Suspect your data

Anything to good or too surprising to be true has a high chance of being not true indeed. Once one accumulates enough domain knowledge and enough feedback from the domain experts, it is time for the more mechanical part of the data analysis phase. In this phase, one can inspect the data manually by looking at e.g. the min-max values of each feature or by plotting the values of each individual feature value through -say- a box-plot. This sometimes works like a charm to identify any errors in the data collection. It also can hint which instances are likely to be outliers.

It is also good to try to interpret the data, in other words what type of a story does the data tell? John Rauser of Amazon points out this issue in his 2011 talk at the Strata conference [66]. He mentions it as the "curiosity" to question the observed effects from the data and attributes it as one of the key characteristics that a person dealing with data should have. Also note that all the prior anomalies could have been simply ignored, unless we suspected the collected data.

4.4 Principle #4, Data collection is Cyclic

The data related war stories and ideas until now were presented in a certain order, yet note that the principles associated with data collection are not necessarily sequential. In other words, they are intermingled and can appear in any order multiple number of times. This fact requires one to get more domain feedback or expert input in the case of data anomalies, then updating the collected data as well as the analysis.

Fayyad et al. provide an in depth discussion of about data collection in the domain of knowledge discovery in databases (KDD) [65]. The recommended framework for KDD by Fayyad et al. is provided in Figure 4.1. Assuming that the principles recommended until now correspond to different nodes of the framework in Figure 4.1, the *"data collection is cyclic"* principle makes the following addition: Every step is connected to all the other steps with a bi-directional edge.

A recent work on the importance of the data collection in the general domain of SE was presented by Menzies et al [5], where they also agree on the fact that data collection has a cyclic nature. In [5], Menzies et al. provide a manifesto for the so called *"inductive engineers"*, who are the engineers that are expected to deal with data from the mining till making sense out of it. They also provide an update of Figure 4.1, where they provide improvements along with the cyclic nature of the data collection.

Note that the data collection related principles are not the fundamental to this thesis or to the presented results later on in this document. They merely serve the purpose of sharing our hands

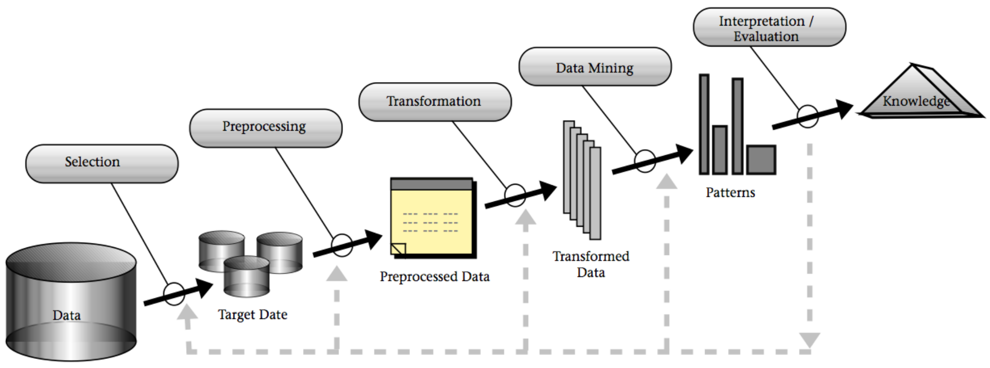


Figure 4.1: The framework given by Fayyad et al. [65].

on experience and some of the war stories that helped us in understanding the data collection. The reason for sharing such experience is the hope that they could help the SEE practitioners in case they also require to collect data.

Chapter 5

Principle #5, Use a Ranking Stability Indicator

Being able to choose the most appropriate software development effort predictor for the local software projects remains elusive for many project managers and researchers. For decades, researchers have been seeking for the "best" software effort predictor. At the time of writing, there is no such a commonly agreed "best" predictor found, which provides consistently the most accurate estimate. The usual conclusion from studies is that different researchers offer conflicting rankings as to what is "best" [4, 14]. It seems, given different historical datasets, different sets of best effort predictors exist under various different situations.

Being able to compare and determine the best effort predictor for different scenarios is critically important to the relevance of the estimates to the target problem. Software effort estimation research focuses on the *learner* used to generate the estimate (e.g. linear regression, neural nets, etc.) in many cases, overlooking the importance of the quality and characteristics of the *data* being used in the estimation process. We argue that this approach is somewhat misguided since, as demonstrated in the following sections of this chapter, learner performance is greatly influenced by the data preprocessing and the datasets being used to evaluate the learner. A combination of a preprocessor and a learner *forms* a complete effort estimation solo-method in general; e.g. the data normalization technique as a preprocessor with linear regression as the learner. The term *"solo-method"* will be used extensively in this and also the following chapter. The reason of calling it *"solo"* is due to the fact that a solo-method contains a single learner. We will see in the following chapter that it is possible to combine different learners. Such combined methods of more than one learner will be referred as *"multi-methods."*

Ranking stability in software effort estimation is of the primary research focus of this chapter. Being able to correctly classify the characteristics of each method allows the most suitable predictors to be used in the estimation process. The study presented in this chapter is a natural result of prior research in SEE, it is based on the success of a previous study described in Menzies et al. [67], where a large number of predictors were applied on COCOMO datasets, and they were able to derive precise and stable ranking of all the predictors under changing parameters in the random number seeds, different evaluation criteria and subsets of the data used.

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

Method combinations can produce vastly different results, in all, this study applies 90 predictors (9 preprocessors $\times$ 10 learners) to 20 datasets and measure their performance using seven performance criteria subject to a statistical check via Wilcoxon non-parametric statistical test. The statistical test is used to generate the so called *win-tie-loss statistics.* One result of exploring such a large space of data and algorithms is that we are able to report stable conclusions. We will refer to our results coming from the *aggregate* of 90 predictors, 20 data sets and 7 error measures as the *"aggregate results."* The aggregate results are validated through focusing on a specific case of a single error measure and a statistical method other than win-tie-loss statistics. For the error measure we chose to use magnitude of relative error (MRE) as it is one of the mostly used performance measures in software effort estimation. As for the statistical method, we developed and applied an algorithm that is called the *"CLUSTER,"* whose details are given in Section 5.4.1. *CLUSTER* first sorts 90 predictors from best to worst (according to median MRE, i.e. MdMRE) and allows each data set to group 90 predictors into a number of clusters. The methods in every cluster are statistically the same according to Kruskal-Wallis statistical test (an ANOVA alternative) and the neighboring methods in consecutive clusters (i.e. the methods where two cluster became disjoint) are statistically different from one another according to Wilcoxon statistical test. So, *CLUSTER* uses:

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

This sanity check showed us that:

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

5.1 Comparison of Multiple SEE Methods

With the availability of different SEE methods, it is becoming a more non-trivial task to select the most appropriate modeling methods for a particular software development situation. Despite decades of research, there is still no consensus on which effort predictors are better or worse than others. Researchers have expressed concerns and even doubt that such a ranking of predictors can ever be generated. For example, Shepperd and Kododa [4] compared regression, rule induction, nearest neighbor and neural nets, in an attempt to explore the relationship between accuracy, choice of prediction system, and different dataset characteristics by using a simulation study based on artificial datasets. A number of conflicting results exist in the literature as to which method provides superior prediction accuracy, and possible explanations are offered including the misuse of an evaluation criteria such as MMRE or a malformed dataset being used etc. All of these can have a strong influence on the relative effectiveness of different prediction models. The conclusion of of Shepperd et al.'s study based on simulation is that it is generally *infeasible* to determine which prediction technique is the "best":

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

5.2 Ranking Instability

More recent results suggest that we should be revisiting the changes of method rankings. Menzies et al. [67] applied 158 predictors to various subsets of two COCOMO datasets. In a result consistent with Shepperd and Kododa, they found the precise ranking of the 158 predictors *changed* according to

- the random number seeds used to generate train/test sets;
- the performance evaluation criteria used;
- and which subset of the data was used.

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

5.3 Experimental Design

The number of all the configurations of different SEE methods can easily exceed thousands. For example, Keung et al. [49] shows that the number of options associated with only CBR methods can easily exceed tens of thousands of different variants. It is impractical to explore all the variants variants of every different SEE methods. Hence, we elected to explore variants commonly used in the literature. For example, we explore neural nets, regression, and analogy because those methods were explored by Shepherd and Kododa [4]. Nevertheless, it is important to note that our conclusions come from the predictors, performance criteria and datasets used in this research. Further work may be required to confirm our findings on other predictors, performance criteria, datasets. The total list of pre-processors and learners used in the experimentation of this chapter has been provided earlier in §2.4.

5.3.1 Experimental Procedure

This research reused the experimental procedure of a recent prominent study [68]. In the leave-one-out experiment, given N projects, 1 project at a time is selected as the test and the remaining $N - 1$ projects are used for training, so eventually we have N predictions (this procedure refers to Jack-knifing in statistics). The resulting N predictions are then used to compute our seven evaluation criteria given in §2.3. To compare the performance of one predictor versus the rest, we used a Wilcoxon non-parametric statistical hypothesis test. Wilcoxon is used to create the win, tie and loss statistics in accordance with the algorithm, whose pseudo-code is given in Figure 2.5.

5.4 Results

After applying leave-one-out to all 20 data sets, the procedure of Figure Figure 2.5 was repeated seven times (once for MAR, MMRE, MMER, MBRE, MIBRE, MdMRE and PRED(25)). Our ninety predictors were then sorted by their total number of losses over all datasets. The resulting sort order is shown in Figure 5.1. The predictor, with fewest losses (**norm/CART(yes)**) was ranked #1 and the predictor with the most losses (**PCA/LReg**) was ranked #90.

Given 89 comparisons and seven performance measures and 20 datasets, the maximum number of losses for any predictor was $89 \times 7 \times 20 = 12,460$. Figure 5.2 sorts all 90 predictors according to their total losses seen in all seven performance criteria (expressed as a percentage of 12,460). The *x-index* of that figure corresponds to the ranks of Figure 5.1; e.g. the top ranked predictor of **norm/CART(yes)** lost in nearly zero percent of our experiments.

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

Figure 5.1: Detailed pre-processor and algorithm combinations (i.e. predictors), sorted by the sum of their losses seen in all performance measures and all public data sets of Figure 2.1. The predictor with fewest losses is ranked #1 and is **norm/CART(yes)**. At the other end of the scale, the predictor with the most losses is ranked #90 and is **PCA/LReg**.

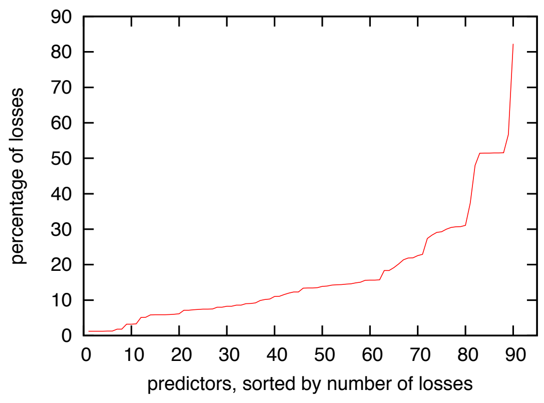


Figure 5.2: The ninety predictors of Figure 5.1, sorted by their percentage of maximum possible losses (so 100% = 12,460).

Figure 5.3 tests the stability of the predictors. In this plot, we check if the sort orders are changed by different performance criteria: In Figure 5.3, we report the mean of maximum rank changes for each predictor with respect to their ordering in Figure 5.1.

- Each error measure defines its own ordering of predictors w.r.t. its win, $loss$ or $win - loss$ values.
- Maximum rank change is the maximum absolute difference between either of these orderings.
- Then, mean of maximum rank changes coming from 7 performance measures gives us Figure 5.3.

The sort order on the x-axis of Figure 5.3 was kept the same as the before. A line drawn parallel to x-axis at $y = 10$ gives predictors, whose mean rank change is less/more than 10. See in Figure 5.3 that $y = 10$ line divides all predictors into 3 regions: a (from predictor 1 to 13), b (from predictor 14 to 64) and c (from predictor 65 to 90). Regions a and c show *"high-ranked"* and *"low-ranked"* predictors respectively. None of the predictors in regions a and c exceed mean rank change of 10, i.e. they are *"stable"* in high and low ranks. In region b *"medium-ranked"* predictors are accumulated. However, all predictors in region b have mean rank changes above 10, i.e. they are *"unstable"* in this region. In a result consistent with prior reports on ranking instability, the lines in each region are not exactly smooth. However, they do closely follow the same general trends as Figure 5.2.

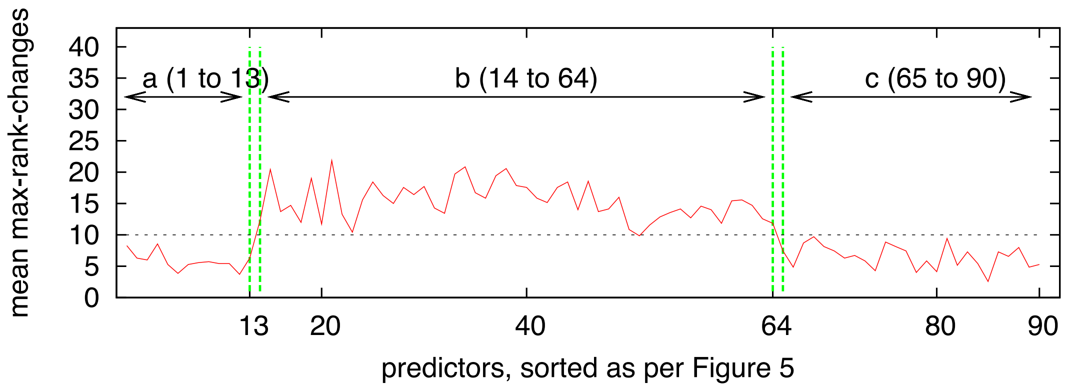


Figure 5.3: Predictors and the mean of their maximum rank changes over all performance measures. Mean rank change of smaller than 10 divides 90 predictors into 3 regions. Region "a" consists of high-ranked stable predictors, whereas region "c" contains low-ranked but still stable predictors. Region "b" on the other hand shows middle-ranked and non-stable predictors.

Since the sort orders seen using the number of losses and mean rank changes over seven performance criteria are mostly stable, we use them to draw Figure 5.4. In that figure, each *x,y* position shows the results of 623 comparisons (each predictor compared to 89 others using seven performance measures; $89 \times 7 = 623$). The *y*-axis of that figure shows the 90 predictors sorted in the rank order of Figure 5.1. For example, the top-ranked predictor **norm/CART(yes)** appears at *y*=1; the **log/ABE0-1NN** result appears at *y*=12; the **log/LReg** results appear at *y*=80; and the worst-ranked predictor **PCA/LReg** appears at *y*=90.

In order to discuss which learners/preprocessors are "best", we divide Figure 5.4 into 3 bands of Figure 5.3. We reserve the lowest band from predictor 1 to 13 (containing the "best" predictors) for the region where all predictors have a mean rank change of smaller than 10. Note that predictors in that region almost always lose less than $\frac{1}{8}$th of the time (i.e. the rows $y = 1$ to $y = 13$ that are almost completely yellow in Figure 5.4). In the other bands (boundaried at $y = 14$ to $y = 64$ and $y = 65$ to $y = 90$), predictors lose much more frequently, i.e. behavior of predictors in the loss percentage graph of Figure 5.4 are in agreement with the rank change graph of Figure 5.3.

Figure 5.5 shows the spectrum of PRED(25) values across the 3 bands. As might be expected, the y-axis sort order of Figure 5.5 predicts for estimation accuracy. As we move over the three

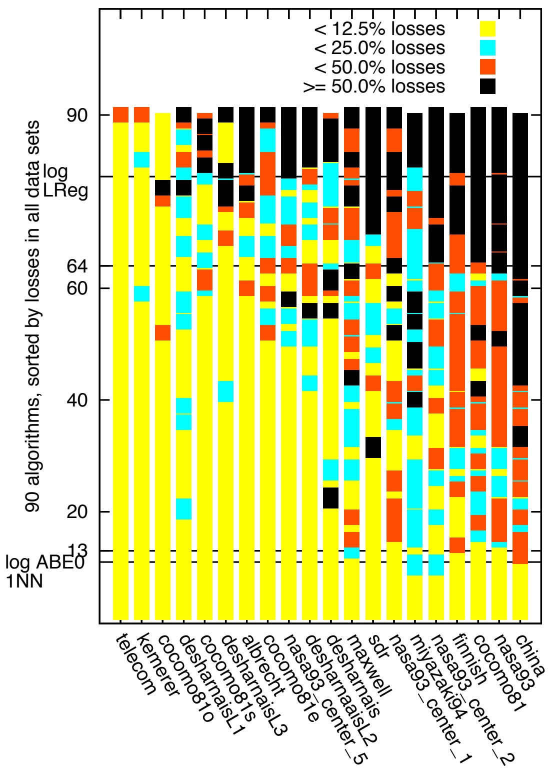


Figure 5.4: Number of losses seen in 90 predictors and 20 datasets expressed as a percentage of the maximum losses possible for one predictor in one dataset (so 100% = *89 comparisonx* × *7 error measures* = 623; 50%=311; 25%=156; 12.5%=78). The predictors on the y-axis are sorted according to Figure 5.1.

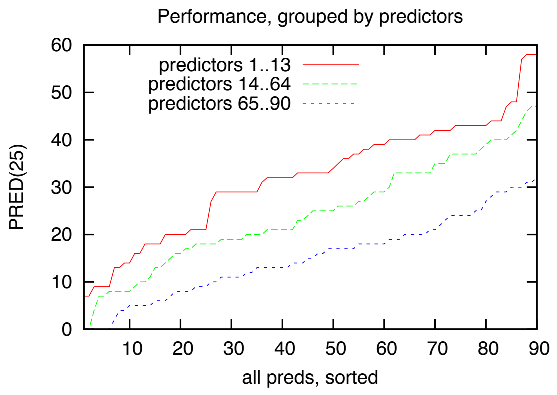


Figure 5.5: Spectrum of Pred(25) across the bands

bands from worst to best, the PRED(25) values double (approximately), thus confirming the unique performance of predictors in each band.

Figure 5.6 shows the frequency counts of preprocessors and learners grouped into the three bands:

- A "good" preprocessor/learner appears often in the lower bands (tendency towards band a). In Figure 5.6, CART is an example of a "good" learner.
- A "poor" preprocessor/learner appears more frequently in the higher bands (tendency towards band c). In Figure 5.6, all the discretization preprocessors (e.g. **freq3bin**) are "poor" preprocessors.
- The gray rows of Figure 5.6 shows preprocessor/learner that are neither "good" nor "poor" (since they exist in all 3 bands and have high frequency counts in bands b and c); e.g. see the **log** preprocessor.

5.4.1 Sanity Check

Our purpose in the sanity check is two-fold:

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

We chose to focus on the specific error measure of MRE. As for the proposed statistical assessment, we devised an algorithm called *CLUSTER*, which makes use of 2 statistical tests in combination: Wilcoxon and Kruskal-Wallis (an ANOVA alternative for the cases where ANOVA's normality assumptions may be invalid). For each data set, *CLUSTER* groups 90 predictors into *"c"* clusters. The predictor(s) grouped in each one of the *"c"* clusters have statistically the same MRE values with one another according to Kruskal-Wallis. The neighboring predictors in two consecutive clusters have statistically different MRE values from one another according to Wilcoxon. The detailed steps of CLUSTER are as follows:

1. Take a single data set D
2. Sort 90 predictors according to their MdMRE values for D in ascending order (i.e. best predictor appears at #1).
3. Set $i = 1$ and $j = 1$
4. Place predictor #i into group #j.
5. Compare MRE values of predictor #i with #i+1 w.r.t. Wilcoxon

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

Figure 5.6: Frequency counts over 7 error measures for preprocessor and learners in the three bands of Figure 5.3.

6. Compare if MRE values of all the predictor(s) in group #j and MRE values of predictor #i+1 w.r.t. Kruskal-Wallis.

7. If both Wilcoxon and Kruskal-Wallis indicate MRE values are statistically the same; then set $i = i + 1$ and go to Step 4; else set $i = i + 1$ and $j = j + 1$ and go to Step 4. Do this until all the 90 predictors are exhausted.

Note that the procedure of *CLUSTER* enables each data set to define its own groups of predictors (clusters), where MRE values of the predictors within each group are statistically the same. The number of groups formed for each data set through the *CLUSTER* as well as the number of predictors appearing in each group are given in Figure 5.7. Except the china data set, all the data sets have less than 6 groups. For the reasons of space, we summed the number of predictors appearing in groups #7 to #16 of china under the *Rest* column. The number distribution of predictors into groups #7 to #16 are: 1, 9, 4, 8, 7, 7, 6, 1, 1, 6 (respectively).

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

Figure 5.7: The number distribution of 90 predictors to the groups formed by *CLUSTER* as well as the Gini indices [69] calculated by that distribution. Data sets are sorted w.r.t. their Gini indices in descending order.

The group counts as well as the distribution of the 90 predictors to these groups for each data set can be used as an indicator of the impurity of the data sets. One of the most commonly used measures of impurity is the Gini index [69], which had also been used as a splitting criterion in the classification and regression trees [48]. The formula for the Gini index of a data set D with *"c"* clusters is given in Equation 5.1, where $|c_i|$ denotes the number of predictors within i^{th} cluster.

$$Gini(D) = 1 - \sum_{i=1}^{c} \left(\frac{|c_i|}{90} \right)^2 \tag{5.1}$$

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

Note that the data sets of Figure 5.7 are sorted according to their Gini indices. Starting from Gini index of 0 we grouped all the 20 data sets into 3 bins with increment of 30 in the Gini index. The resulting bins are called: Diverse (with 6 data sets), semi-diverse (with 8 data sets) and non-diverse (with 6 data sets). These bins are indicated on Figure 5.7. When performing our sanity check, we will focus our attention to diverse and semi-diverse data sets only.

Figure 5.8 is our sanity check on the top 13 learners. It shows the success of top 13 learners in the diverse and semi-diverse data sets. The cells of Figure 5.8 show in which group (top 1, 2 and so on) each one of the top 13 learners appeared. The cells in which top 13 learners do not appear within the best group (i.e. top *"1"*) are highlighted. Note that there are only two data sets for which the majority of the cells are highlighted: china and desharnaisL3. Thinking that china data set has 16 groups and that the top group has only 2 predictors in it, top 13 learners to be in the 2^{nd} best group is not dramatic result either. Among the 14 data sets, there is only 1 contradictory example, where top 13 learners found through an aggregate analysis do not hold for a specific analysis.

Our sanity check on the worst performing (i.e. bottom) 26 learners is given in Figure 5.9, which also only shows the diverse and semi-diverse data sets. Figure 5.9 shows in which group (top 1, 2 and so on) bottom 26 learners appeared. The cells where the bottom 26 learners appear within the best group (i.e. top *"1"*) are highlighted. The data set that contradicts the most with our aggregate

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

Figure 5.8: The number of groups per dataset (only diverse and semi-diverse data sets) and the group-rank of the top 13 learners. The cells where the top 13 predictors do not appear on the top group are highlighted.

conclusions is cocomo81s, which has 11 highlighted cells. For the rest of the data sets, the bottom 26 learners, which were identified through an aggregate analysis also perform poorly in a specific analysis.

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

Dataset	# Groups	log SWR	log PCR	log PLSR	width3bin PLSR	width3bin 1NN	width5bin PCR	norm PCR	width3bin PCR	freq5bin PCR	freq5bin SWR	width3bin LReg	freq3bin PCR	width5bin LReg	freq3bin PLSR	freq5bin PLSR	log LReg	freq3bin SWR	freq5bin LReg	width5bin NNet	norm NNet	width3bin NNet	log NNet	freq3bin NNet	freq5bin NNet	freq3bin LReg	PCA LReg
nasa93	6	5	6	6	5	4	5	5	5	5	6	5	5	5	5	5	5	5	5	5	6	5	6	6	6	5	5
cocomo81	4	4	4	4	3	1	3	3	3	4	4	4	4	3	3	4	4	4	4	4	4	4	4	4	4	3	3
miyazaki94	4	4	4	4	2	3	2	3	2	3	3	4	3	4	3	4	4	3	3	4	4	4	4	4	4	3	2
nasa93center2	3	2	3	3	2	2	3	3	3	3	3	2	3	2	3	3	2	3	2	3	3	3	3	3	3	2	3
maxwell	4	3	3	3	3	3	3	3	3	3	3	3	3	2	3	3	3	2	3	4	4	3	3	4	4	3	3
china	16	13	12	16	11	10	12	11	13	13	13	11	12	10	13	15	13	12	12	16	16	16	14	16	16	12	12
sdr	3	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	2	3	2	2	2	2	2	2	3	3
desharnais	3	2	2	2	1	2	2	2	2	2	2	2	2	1	2	2	2	2	2	3	3	3	3	3	3	2	3
finnish	3	2	2	2	2	2	2	3	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	2	3
cocomo81e	2	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2
nasa93center1	3	1	1	2	2	1	2	2	2	2	2	1	2	1	2	2	2	2	1	3	3	3	3	3	3	2	3
desharnaisL3	4	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	4	2	4	2	2	2	2	2	2	2	4
nasa93center5	2	2	2	2	2	1	1	2	1	1	2	1	1	1	2	2	2	2	2	2	2	2	2	2	2	1	2
cocomo81s	2	2	2	1	2	1	2	2	1	2	1	2	2	2	2	2	1	2	2	1	1	1	1	1	1	2	2

Figure 5.9: The number of groups per dataset and the group-rank of the bottom 26 learners to these groups. The cells where the bottom 26 learners appear on the top group are highlighted.

5.5 Discussion

5.5.1 Findings

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

Finding1: Observing the small amounts of fluctuation (or jitter) in Figure 5.3 we can see that our results may not be considered 100% stable, however they are sufficiently stable to draw conclusions. We conjecture that prior reports on ranking instability could stem from using too few data sets or too few predictors.

Finding2: Figure 5.1 shows the preprocessor and learner combination is important, as their current rank can be changed if a different preprocessor is used in combination with the learner. For example, the top-ranked predictor that uses **CART(yes)**, is driven down to rank 60 if the preprocessor is changed from norm to **freq3bin**. Clearly, the effectiveness of a learner can be significantly altered by seemingly trivial details relating to data preprocessing as it will change the dataset characteristics input to the learner. Hence, in future, researchers should explore learners *and* the preprocessors, as they are both equally important.

Finding3: Observe in Figure 5.6 how **SWR**, **LReg** and **NNet** are stand-out learners that fall entirely into the worst two bands. Proponents of these learners need to defend their value for the purposes of effort estimation.

The relatively poor performance of simple linear regression is a highly significant result. **LReg**, with a log preprocessor, is the core technology of many prior publications; e.g. the entire COCOMO project [20]. Yet as shown in Figure 5.4, w.r.t. loss values over all error measures, **log/LReg** ranks very poorly (position 80 out of a maximum of 90 predictors). We also did experiments at individual level of error measures. At individual level the ranking is not very different either, i.e. the ranking of LReg w.r.t. loss values over MAR, MMRE, MMER, MBRE, MIBRE, MdMRE and Pred(25) are 80, 76, 81, 80, 75, 76 and 78 respectively.

Finding4: While **SWR** falls into the *worst two bands* of the learners, it is most commonly found in the *best two bands* of the preprocessors. That is, stepwise regression is a *poor learner* but a *good preprocessor*. Hence, in future, the fate of **SWR** might be as an assistant to other learners.

Finding5: While simple regression learners like **LReg** are deprecated by this study, more intricate regression learners like regression trees (CART) and partial linear regression **PLSR** are found in the better bands. Hence, in future, proponents of regression for effort estimation might elect to explore more intricate forms of regression than just simple **LReg**.

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

Finding7: Simple predictors (e.g. *k*=1 nearest neighbor on the log of the numerics) perform nearly as well as the top-ranked predictor. Figure 5.10 compares the PRED(25) results between rank=12 and rank=1. The datasets in that figure are sorted by the difference between the top-ranked and the twelfth-ranked predictor. Except for China dataset, the difference in PRED(25) values is either slightly negative, or positive. That is, even though the rank=1 predictor is *relatively* “best” (measured according to our comparative Wilcoxon tests), when measured in an *absolute* sense, it is not impressively better than simpler alternatives.

Finding7 is an important result, for three reasons. Firstly, there are many claims in the literature that software project follows a particular parametric form. For example, in the COCOMO project, that form is $effort \propto KLOC^x$). The fact that non-parametric instance predictors perform nearly as well as our best predictor suggests that debates about the parametric form of effort estimation is misguided. Also, it means that the value of certain commercial estimation tools based on a

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

Figure 5.10: Using all data sets to compare the Pred(25) of **norm/CART(yes)** (rank=1) and *log/ABE0-1NN* (rank=12).

particular parametric form may not be much more than simple instance-based learners.

Secondly, analogy-based estimation methods are widely used [13, 15, 28, 46, 68, 70–74]. Finding7 says that while this approach may not be 100% optimal in all cases, compared to our best predictor found by this study, there is not a dramatic lost if estimates are generated by analogy. Prior to this publication, we are unaware of a large comparative study relating to this matter.

Thirdly it is easier to teach and experiment with simpler predictors (like the **log/ABE0-1NN** predictor at rank=12) than more complex predictors (like the **norm/CART** predictor at rank=1). For example, recently we have been experimenting with a very simple variant of ABE0-1NN that is useful as a learner to find software process change [75]. Such experimentation would have been hindered if we tried to modify the more complex CART learner (particularly if we included sub-tree pruning).

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

5.6 Conclusion

In this chapter we presented the results of a research, whose results come from the combination of 10 learners and 9 data preprocessors into 90 effort estimation predictors. These were applied to 20 public datasets of Figure 2.1. Performance was measured using 7 performance indicators (AR, MRE, MER, MdMRE, MMRE, PRED(25), MBIRE). Performances were compared using a Wilcoxon ranked test (95%). This procedure can be used as a ranking stability indicator for selecting the most suitable effort predictor in SEE, which is an important stage in the estimation process. Eight findings are identified to be noteworthy:

1. Prior reports of ranking instability about effort estimation may have been overly pessimistic. Given relatively large number of publicly available effort estimation datasets, it is now possible to make stable rankings about the relative value of different effort predictors.

2. The effectiveness of a learner used for effort estimation (e.g. regression or analogy methods) can be significantly altered by data preprocessing (e.g. logging all numbers or normalizing them zero to one).

3. Neural nets and simple linear regression perform much worse than other learners such as analogy learners.

4. Stepwise regression was a very useful preprocessor, but surprisingly a poor learner.

5. Non-simple regression methods such as regression trees and partial linear regression are relatively strong performers.

6. Regression trees that use tree pruning performed best of all in this study (with a preprocessor that normalized the numerics into the range zero to one).

7. Very simple predictors (e.g. K=1 nearest neighbor on the log of the numerics) performed nearly as well as regression trees.

8. It is important to validate the stable conclusions derived from aggregate results through specific scenarios. Such a validation in this study showed that stable conclusions hold for the specific scenarios. Yet, it is still a possibility that *"rarely"* the aggregate and specific results will favor different predictors.

The afore-mentioned findings are the result of an investigation for ranking deltas of a large set of SEE solo-methods executed on a large space of 20 data sets and 7 error measures. Such use of rank delta analysis can help to identify the superior solo-methods, which consistently outperform inferior solo-methods. Knowledge of superior solo-methods (as we will see in the next chapter) can be used to propose alternative ideas (e.g. ensembles of solo-methods). Hence, the principle recommended as a result of this chapter is:

Principle #5: Use a Ranking Stability Indicator

The contributions associated with the research presented in this chapter are:

- A method to identify successful methods using their rank changes.
- An evaluation method of the diversity of the SEE data sets.
- Proof that use of a ranking stability indicator can help to identify successful methods both for aggregate and specific cases.

Chapter 6

Principle #6, Assemble Superior Methods

The results presented in this chapter build upon the results of the prior chapter. In the previous chapter, we observed that the rank of a method is bound to change when the data set or the error measure changes. Yet we observed that the the amount of rank change (i.e. δr) is much smaller for certain methods (the so called superior solo-methods). In this chapter, we exploit this fact to combine superior solo-methods (a learner augmented with a pre or post-processing step) with low δr values into multi-methods. A multi-method is a combination of two or more solo-methods. When the performance of the multi-methods are compared to those of the solo-methods, we are able to observe that multi-methods are consistently more successful than the solo-methods.

This should not be a surprising result. Many researchers argue that, in theory, best estimates come from combinations of multiple predictions. For example, Jorgensen advises that, for expert-based estimation, it is best to generate estimates from multiple methods [27]. Machine learning community also shares this optimism regarding multi-methods. For example Seni et al. report that taking the average of the estimates coming from multiple solo-methods yield better results than the individual solo-methods [76]. Similar conclusions are offered by other researchers in the field of statistics [77] and machine learning [78, 79].

Unlike the studies in machine learning and statistics, the SEE studies that dealt with multi-methods are pessimistic about their performance [9, 10, 61].

- Kocaguneli et al. failed to improve estimation through averaging the predictions of 14 estimators [10].

- Baker could not improve estimation accuracy through boosting [61].

When compared with the previous ensemble results in SEE, the results presented in this chapter regarding ensembles turn out to be different. Whereas previous studies on the ensembles (through different strategies) were reporting that ensembles are not statistically better than single learners, our study reports that (through the right strategy) ensembles can outperform single learners. The difference between the prior work and this one is that they were based on the assumption that all the solo-methods were candidates to be included in multi-methods. However, this is hardly the case. We know from the previous chapter that only a minority of the solo-methods have low rank changes and successful performance. Hence, in this chapter we will recommend that only the *superior* solo-methods should be employed in multi-methods.

The results of this chapter resolve the contradiction between theory and experimental results in SEE. To the best of our knowledge, this is the first such result in the effort estimation literature that is supported by extensive experimentation. We therefore offer the following advice for a successful ensemble of methods:

1. Try a large number of methods among which there are at least some good methods (shown to have a good performance by prior work).

2. Sort the methods using the evaluation methods discussed in this document. Discard all but the best solo methods.

3. Built ensembles from the remaining solo-methods.

6.1 Ensemble of Methods

A standard machine learning technique is to try multiple methods on the available data, then recommend the one that performs best [8]. Many effort estimation papers apply this technique to demonstrate that (say) their preferred new method is superior to those proposed in prior work.

Ensemble learning takes a different approach. Rather than choosing *one* method, ensembles build multiple predictors, where estimates coming from different learners are combined through particular mechanisms, e.g. voting of individual learner estimates on the final prediction [80]. Before continuing any further we need to clear a terminology difference. From now on the term *"learner"* refers to a stand-alone machine learning algorithm without any supplemental pre or post processing step (e.g. k-NN, neural nets etc.), whereas the term *"solo-method"* will refer to a machine learning algorithm supplemented with a pre-processing option (e.g. logging+k-NN, discretization+neural nets etc.).

Ensembles are useful since any particular learner comes with its own assumptions [8]. These assumptions may be best suited to different parts of the training data [6, 8, 80]. In ensembles, methods can augment each other, i.e. a method patches errors made by another method. For example, when reducing estimated mean-squared-error, multi-methods attain smaller or equal error rates than single-methods [76].

It is a recommended practice to combine solo-methods that have different characteristics [6–8]. There are many techniques to attain different-characteristic multi-methods. The first way is through representation of the data. The multi-method structure may be based on uni-representation (all learners use same representation of data) or multi-representation (different learners use different representations) [8]. Examples to such strategies are the use of different feature sets [81, 82] or different training sets [83].

The second way is through architectural methodologies. Bagging (Bootstrap Aggregating) and boosting are among the most common examples of that approach [8, 84]. In bagging n-many solo-methods are independently applied on n-many different training samples, where each training sample is selected via bootstrap sampling [8] with replacement. Boosting on the other hand arranges solo-methods in a sequential manner: each solo-method pays more attention to the instances on which previous method was unsuccessful. Among the reported results, boosting is mostly reported to be considerably better than bagging [84–86], but has trouble in handling noisy datasets [84, 86].

The ensembles are employed in various domains. For example biometrics domain has very successful applications in terms of bringing together the information coming from different sources. Ross et al. show in [87] a successful application for an ensemble of 3 different biometrics modalities. In [88] Ross et al. perform ensembles at feature extraction level. Another good domain for the use of ensembles of learners is the SE domain. The details of the studies dealing with ensembles in SE are given in the following subsection. However, we suffice to note that there is a slight terminology difference between the two domains. The terminology used in biometrics domain [87] for the ensemble of biometric systems is *"fusion"*. On the other hand, in SE domain the term used for combining different learners or algorithms is *"ensemble"*. The SE community prefers to stick to conventional names. For example a criticism to an earlier work of us [10] was *"using unorthodox terms"* for the SE domain, when dealing with ensembles. Therefore, we will use the term *ensemble* when referring to combination of learner/algorithms throughout this report.

6.2 Ensemble of Methods in SE

Ensemble of solo-methods into multi-methods are widely used in data mining [76]. On the other hand, the success of multi-methods in one domain does not promise that they will be successful in another domain with different types of problems. For example in SE domain, they are reported to be unsuccessful. Khosgoftaar et al. [9] question the performance of different multi-method schemes under different scenarios in the domain of software quality. They use different combinations of 17 learners induced on 7 datasets and report that multi-methods induced on single datasets do not yield a significant increase in prediction accuracy.

Kocaguneli et al. [10] replicated [9] in the domain of software effort estimation. They exploited combination of 14 methods applied on 3 software effort estimation datasets. Their conclusion was similar to that of [9]: the application of multi-methods under different scenarios did not provide a significant increase in the estimation accuracy. Similarly, in [11], different learners were employed in two types of committees, but only one of them was reported to be successful.

Another example to ensembles is the committee of *single-type* learners, where multiple versions of a single algorithm are combined. Pahariya et al. use linear combinations of genetic algorithms in [89], where they report improvements over single-learners. In [90] Kultur et al. report improvements through collections of neural networks. Note that such single-learner methods fall into the category of solo-methods (since there is only one type of learner) in this work.

There are also some applications of ensemble methods in effort estimation so as to process datasets: In [91], Twala et al. use multiple imputation techniques to handle missing data and in [92] Khoshgoftaar et al. make use of learner ensembles as a filter to improve the data quality.

Our ensemble is different from the above. We take care to prune inferior solo methods *before* building the ensemble. As shown below, this leads to quite successful effort estimators.

6.3 Solo and Multi Methods

The effort estimation methods studied in this chapter fall into two groups: *solo*-methods and *multi*-methods. *Solo*-methods are some combination of a *pre-processing option* and a *learner*. For example, Boehm's preferred effort estimation method uses a log transform as the pre-processing option, then linear regression as the learner [20]. *Multi*-methods are combinations of at least two solo-methods.

6.3.1 Multi-Methods

Multi-methods combine two or more solo-methods. Many combination schemes have been proposed in the literature [1, 6–8]. Complex combination schemes include bagging [69], boosting [93] or random forests [94, 95]. Simpler methods include computing the mean, median or inverse-ranked weighted mean (IRWM [1], see Figure 6.1) of estimates coming from *n-many* solo-methods.

In IRWM, the final estimates from M methods $e_1, e_2, ..., e_m$ that have been ranked $r_1, r_2, ..r_m$ is a weighted sum by the ranks of all methods in the ensemble. The top and bottom-ranked methods of m methods get a weight of m and 1 (respectively). More generally, a method with rank r_i gets a weight of $m + 1 - r_i$. The final estimate in IRWM is hence $(\sum_i (m + 1 - r_i) \cdot e_i)$ / $(\sum_i i)$.

Figure 6.1: IRWM. Generalized from [1].

Our aim is not to investigate complex schemes, but to observe how multi-methods perform compared to solo-methods on effort datasets. Therefore, we adopt simple schemes (mean, median and IRWM). In the ideal case, different multi-methods would perform optimally under different combination schemes. For example when combining a high number of solo-methods, it would be better to use a scheme that would catch the central tendency and be able to handle extreme values, e.g. median. However, our findings do not support that implication. In other words, our best performing multi-method includes a high number of solo-methods and its combination scheme is IRWM. That discrepancy between implications and results is a familiar concept in forecasting literature: It is difficult to give robust guidelines as to combine the methods in an optimum way [96].

Investigation of robust combination schemes and their implications would be a good future direction to our study.

6.3.2 90 Solo-Methods

In our experiments, we used 10 different pre-processing options and 9 learners, which were introduced in §2.4. As one would remember from that chapter, the combination of 10 pre-processing options and 9 learners results in 10*9=90 solo-methods.

Note that new methods are being constantly invented (e.g. see [40, 46, 90]) so we make no claim that the 90 methods explored here cover the space of all possible effort estimators. Hence, we take care *not* to conclude that some particular combination of solo-methods is the best. Rather our conclusions will be that, given a set of models, it is best to rank them and generate multi-model estimators from the top-ranked models.

We also make no claim that one study can cover all the pre-processing options there is in the literature and our study is is no exception. Our study covers a total of 10 options for pre-processing, however it does not cover the effects of noise or outlier removal. The investigation of the effects of noise/outlier removal on method performance would in fact be an interesting future direction to this study.

6.3.3 Experimental Conditions

One important point highlighted in the studies of Shepperd et al. [4] and Menzies et al. [67] is to evaluate the methods subject to various experimental conditions, since the change in experimental conditions are likely to change the ordering of methods. The experimental conditions, which are reported to be particularly important are [4, 67]:

1. *Performance measures* that measure a method's performance,
2. *Summary* over multiple datasets and performance measures,
3. *Data sets*.

For our experiments, we use 7 different error measures: MAR, MMRE, MdMRE, MMER, PRED(25), MBRE, MIBRE. For the summary over multiple performance measures and datasets we use the following procedure: Each of our 90 methods were compared to 89 others using the procedure of Figure 2.5. According to this procedure, we sum the win, $loss$, $win - loss$ values coming from and we compare our methods according to each sum separately. These comparisons can be summarized through many ways:

1. Number of losses;
2. Number of wins;
3. Number of wins-losses

In other words, for a method to be highly ranked, it must perform well across *all* error measures. The δr ,i.e. the rank change amount of a method is calculated in the same manner as introduced in §5.4 of Chapter 5. Eventually, we run this analysis over 20 public data sets of Figure 2.1. Note that the results of this chapter are an extensive analysis of different conditions for effort estimation experiments. Given 89 comparisons among solo-methods, 7 error measures, and 20 datasets, then each method appears in $89 \times 7 \times 20 = 12,460$ comparisons.

6.4 Methodology

6.4.1 Focus on Superior Methods

Figure 6.2 is a plot, which was introduced in Chapter 5, that shows the success of the solo-methods through ranking and the variability in that ranking. The x-axis of Figure 6.2 shows the the ranking of the 90 solo-methods, according to number of losses over all 7 error measures and 20 data sets. The most successful methods have the lowest number of total losses whereas the the least successful ones have the highest number of total losses. Then solo-methods are ranked on the x-axis starting from the best one. Therefore, better methods appear on the left-hand-side of that figure (so the top-ranked method appears at position $x = 1$).

The ranking of methods is identified by the x-axis of Figure 6.2, whereas the variability in that ranking is given by the y-axis. The y-axis of Figure 6.2 shows the maximum *changes*, δr, seen for each method as we compare the ranks across number of losses, number of wins, and number of wins-losses. In other words, a solo-method has different rankings according to its win, $loss$

or $win - loss$ values and the related number seen on the y-axis is the biggest difference between its best and worst rankings. In a result consistent with Shepperd et al., all methods have $\delta r > 0$. However, the good news is that the top-ranked methods have a very low δr. That is, even if these top-ranked methods jumped rank by their maximum δr, then they would still be performing better than most of the other 90 methods.

In signal processing, it is standard practice to segment data based on the region of maximal change [97]. An inspection of Figure 6.2 shows that the region of maximal δr occurs after $X = 13$. This is an interesting division since the region $1 \leq X \leq 13$ contains methods with high rank and low δr. We call these methods *superior* and the rest *inferior*. The list of top 13 methods are shown in Table 6.1.

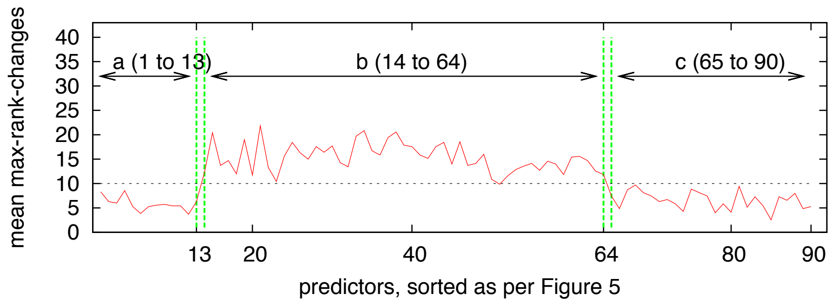


Figure 6.2: Methods and the associated changes, i.e. δr values. Note the sudden increase in δr values, after $X = 13$. We call methods in the region $1 \leq X \leq 13$ as *superior*.

Table 6.1: Ranking of top-13 *superior* solo-methods and related δr values. These solo-methods are combined in various ways to form 12 multi-methods.

rank	δr	pre-processing option	learner
1	8	norm	CART (yes)
2	6	norm	CART (no)
3	6	none	CART (yes)
4	9	none	CART (no)
5	5	log	CART (yes)
6	4	log	CART (no)
7	5	SWR	CART (yes)
8	6	SWR	CART (no)
9	6	SFS	CART (yes)
10	5	SFS	CART (no)
11	5	SWR	ABE0-1NN
12	4	log	ABE0-1NN
13	5	SWR	ABE0-5NN

The names of the top 13 *superior* solo-methods of Figure 6.2 are listed in Table 6.1. When we look at Table 6.1, we see that none of the superior solo-methods try to fit one model to all the data:

- The CART regression tree learner appears at ranks 1 through 10 of Table 6.1. Each branch of a regression tree defines one context in which an estimate may be different.
- Analogy-based estimation (ABE) appears at ranks 11,12,13. ABE builds a different model for each test instance (using the test instance's k-th nearest neighbors).

Solo-methods are not the focus of this report. Hence, we move on to discuss multi-methods.

6.4.2 Build Ensembles

To form multi-methods, we build ensembles using the top M solo-methods in the sort order of Table 6.1. In this study, we use $M \in \{2, 4, 8, 13\}$.

To generate an estimate, we ask all members of an ensemble to offer a prediction. These are combined in one of three ways: mean, median and IRWM.

With this scheme, we have *4 groups of solo-methods (group of top 2 methods, another group of top 4 methods, then group of top 8 methods and finally the group of top 13 methods) times 3 (mean, median, IRWM) = 12 multi-methods*. These multi-methods are then ranked along-side the solo-methods in the same manner as Figure 6.2. This gives us the comparison of *90 solo-methods + 12 multi-methods = 102 methods*. Every method is compared to 101 others with respect to seven error measures and over 20 datasets. Therefore, the maximum number of comparisons for any method now becomes $101 \times 7 \times 20 = 14,140$. To the best of our knowledge this is the most extensive effort estimation experiment yet reported in the literature (and for extensive non-experimental studies, see [3,64]).

6.5 Results

Figure 6.3 shows the rank of our 102 methods. As before, the x-axis ranks the methods according to number of losses and the y-axis shows the δr of each method. Table 6.2 shows the 102 methods, sorted in the same way as the x-axis of Figure 6.3.

Two aspects of these result are worth commenting:

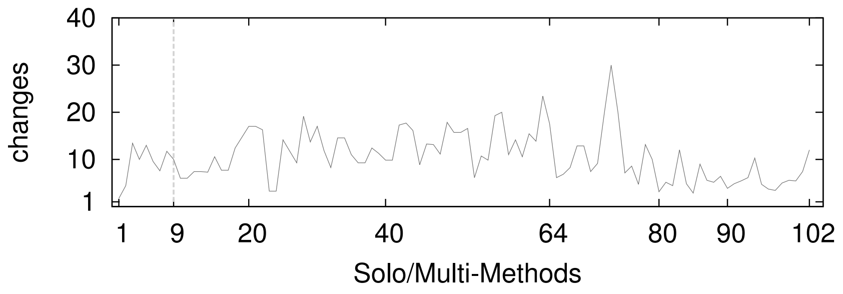


Figure 6.3: Rank changes of solo and multi-methods. Region $1 \leq X \leq 9$ contains 9 out of 12 multi-methods. See that **Top13/Irwm** at $X = 1$ has a δr of 1, i.e. it outperforms all other methods w.r.t. 7 different error measures and 20 datasets.

- The top $X = 9$ methods (marked by a dashed line) are all multi-methods. The remaining multiple methods appear at ranks 14,15,18. That is, in the majority case ($\frac{9}{12} = 75\%$), combinations of methods perform better that any solo method. Further, in all cases ($\frac{12}{12} = 100\%$), they are ranked higher than the majority of other methods.

- The multi-method at $X = 1$ also has the lowest δr of any method in this study. This method generated estimates using the mean value of 13 top-ranked methods.

Note that the second result is exactly the "ensembles are better" result as might be predicted by the researchers mentioned at the beginning paragraphs of this chapter [27, 76–79]. We remind that prior SE researchers who failed to find that "ensembles are better" did not prune away inferior solo-methods before building an ensemble.

Better yet, as shown in Figure 6.3, this largest ensemble at rank $X = 1$ had the lowest δr seen in any of our 102 methods ($\delta r = 1$). This result underscores the main message of this report: the method that scored the best (and had the greatest stability across different experimental conditions) was the one that used the most number of *superior* solo-methods.

Figure 6.4 shows the sum of win, tie and $loss$ values for the methods of Figure 6.3. Every method of Figure 6.3 is compared to 101 other methods, over 7 error measures and 20 datasets, so the maximum value that either one of the win, tie, $loss$ statistics can attain is: $101 \times 7 \times 20 = 14,140$. Note in Figure 6.4 that except the low performing methods on the right hand-side, the tie values are in $10,000 - 12,000$ band. Therefore, they would not be so informative as to differentiate the methods, so we consult to win and $loss$ statistics. There is a considerable difference between

Table 6.2: Detailed pre-processing option and learner combinations and related δr values. Methods are sorted by the sum of their losses seen in all performance measures and all data sets. The method with fewest losses is ranked #1 and is **Top13/Mean**. At the other end of the scale, the method with the most losses is ranked #102 and is **PCA/LReg**.

rank	δr	pre-proc.	learner	rank	δr	learner	
1	1	Top13	Irwm	52	17	norm	SWReg
2	4	Top13	Mean	53	6	none	SWReg
3	13	Top13	Median	54	11	freq3bin	ABE0
4	10	Top2	Mean	55	10	width3bin	CART (yes)
5	13	Top4	Mean	56	19	width3bin	CART (no)
6	10	Top2	Median	57	20	PCA	ABE0
7	8	Top4	Median	58	11	PCA	NNet
8	12	Top8	Median	59	14	none	NNet
9	10	Top2	Irwm	60	11	width5bin	SWReg
10	6	Top4	Irwm	61	15	width3bin	ABE0
11	6	norm	CART (yes)	62	14	SWReg	NNet
12	7	norm	CART (no)	63	23	SFS	NNet
13	7	none	CART (yes)	64	18	width5bin	1NN
14	7	none	CART (no)	65	6	SWReg	SLReg
15	11	Top8	Irwm	66	7	none	SLReg
16	8	log	CART (yes)	67	8	norm	PLSR
17	8	log	CART (no)	68	13	width5bin	ABE0
18	12	Top8	Mean	69	13	norm	SLReg
19	15	SWReg	CART (yes)	70	7	freq5bin	1NN
20	17	SWReg	CART (no)	71	9	freq3bin	CART (yes)
21	17	SWReg	1NN	72	20	freq3bin	CART (no)
22	16	SFS	CART (yes)	73	30	PCA	1NN
23	3	SFS	CART (no)	74	20	freq3bin	1NN
24	3	log	1NN	75	7	width3bin	SWReg
25	14	SWReg	ABE0	76	9	log	SWReg
26	12	PCA	PLSR	77	5	width5bin	PLSR
27	9	none	PLSR	78	13	log	PCR
28	19	SWReg	PCR	79	10	log	PLSR
29	14	PCA	PCR	80	3	width3bin	1NN
30	17	none	PCR	81	5	width3bin	PLSR
31	12	SFS	1NN	82	4	width5bin	PCR
32	8	PCA	CART (yes)	83	12	norm	PCR
33	15	PCA	CART (no)	84	5	width3bin	SLReg
34	15	SFS	ABE0	85	3	width3bin	PCR
35	11	norm	1NN	86	9	freq5bin	PCR
36	9	none	1NN	87	6	freq5bin	SWReg
37	9	freq5bin	CART (yes)	88	5	width5bin	SLReg
38	12	freq5bin	CART (no)	89	6	freq3bin	PCR
39	11	freq5bin	ABE0	90	4	freq3bin	PLSR
40	10	SFS	SLReg	91	5	freq5bin	PLSR
41	10	width5bin	CART (yes)	92	5	log	SLReg
42	17	width5bin	CART (no)	93	6	freq3bin	SWReg
43	18	SWReg	PLSR	94	10	freq5bin	SLReg
44	16	SFS	PLSR	95	5	width5bin	NNet
45	9	SFS	PCR	96	4	norm	NNet
46	13	norm	ABE0	97	3	width3bin	NNet
47	13	PCA	SWReg	98	5	log	NNet
48	11	none	ABE0	99	6	freq3bin	NNet
49	18	SWReg	SWReg	100	5	freq5bin	NNet
50	16	log	ABE0	101	7	freq3bin	SLReg
51	16	SFS	SWReg	102	12	PCA	SLReg

the best and the worst methods in terms of win and $loss$ values (in the extreme case it is close to $4,000$). In a way Figure 6.4 is a sanity check of Figure 6.3, because it shows that the rankings reported in Figure 6.3 are due to considerable win and $loss$ value differences between high (left

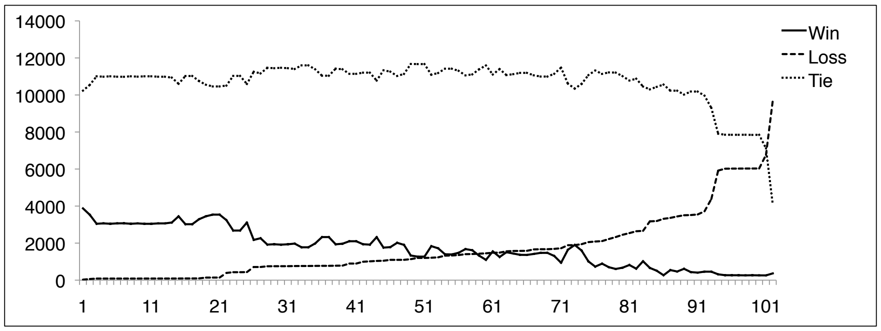


Figure 6.4: The sum of win, tie and loss values for all methods of Figure 6.3 (over all error measures and all datasets). Since one method is compared to 101 other methods, over 7 error measures and 20 datasets, the sum of win, tie and loss values is: $101 \times 7 \times 20 = 14,140$.

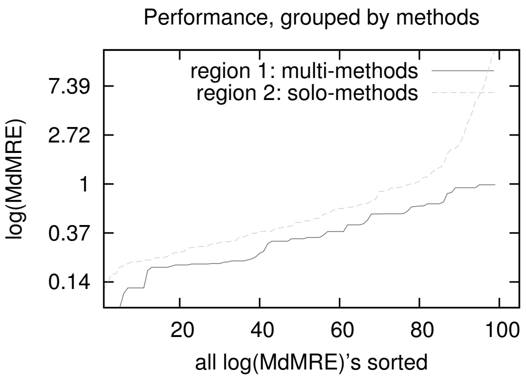


Figure 6.5: Spectrum of MdMRE values for 2 regions of methods: Solo-methods and multi-methods. Here we keep the order of methods same as those given in Table 6.2 and divide those methods into 2 regions. Notice how multi-methods attain the lowest MdMRE scores.

hand-side) and low (right hand-side) performing methods.

Other results offer yet more evidence for the superiority of multi-methods over solo-methods. Figure 6.5 sorts the MdMRE values of all the solo-methods and all the multi-methods:

- Multi-methods generated lower MdMRE values than the solo-methods;
- While some of the solo-methods have very large errors (as observed from the steep right-hand-side of the dashed line in that figure), note that *none* of the 12 best multi-methods have large MdMRE values.

That is, multi-methods are far less prone to incorrect effort estimates than solo-methods.

6.6 Discussion

6.6.1 Learning Curve

One of our observations is that ensembles are more trustworthy (less ranking instability). For someone with a strong background in machine learning algorithms, the number of learners to combine is not an issue as he/she has already gone through the learning curve. However, from a practitioner's point of view, there is a cost-benefit trade-off between:

- the cost of learning new learners
- and the additional performance benefit

We acknowledge the fact that building an ensemble model from scratch may be too challenging for practitioners without prior machine learning experience. However, our industry-collaborated project experience shows that once the model is built by researchers, its adoption/implementation by practitioners is a pretty straightforward process [98–101]. This section provides an in depth discussion and alternative solutions for such practitioners.

Table 6.3: The learning curve of the top-13 *superior* solo-methods. The column *"# of additions"* shows how many algorithms (learner or pre-processor) are needed for the transition to current row from the previous one and the required algorithm for that transition is given in brackets. The column *"total learned" shows how many algorithms are needed to be learned until the current row. Assumption is that common numeric methods such as normalizing and taking the natural logarithm of numbers is known in advance.*

rank	pre-processor	learner	# of additions	total learned
1	norm	CART (yes)	1 [CART (yes)]	1
2	norm	CART (no)	1 [CART (no)]	2
3	none	CART (yes)	0	2
4	none	CART (no)	0	2
5	log	CART (yes)	0	2
6	log	CART (no)	0	2
7	SWR	CART (yes)	1 [SWR]	3
8	SWR	CART (no)	0	3
9	SFS	CART (yes)	1 [SFS]	4
10	SFS	CART (no)	0	4
11	SWR	ABE0-1NN	1 [ABE0-1NN]	5
12	log	ABE0-1NN	0	5
13	SWR	ABE0-5NN	0	5

The best multi-method reported in this report requires combination of top-13 solo methods. The learning curve associated with top-13 solo methods is given in Table 6.6.1. From Table 6.6.1

we see that a practitioner willing to use multi-methods at all has to learn at least 2 learners (CART(yes) and CART(no)). Assuming that this hypothetical practitioner knows common numeric manipulations such as normalizing an array of numbers and taking logarithm, mean or median of these numbers; learning just 2 learners in fact enables him/her to use up until the 6^{th} best performing solo-method. By using top-6 solo methods, we can build the 3^{rd}, 4^{th}, 5^{th} and 6^{th} best methods out of 102 methods in Table 6.2.

In summary, building successful multi-methods is not necessarily a difficult process for a practitioner. See in Table 6.6.1 that building the best method reported here requires learning 5 algorithms (learner or pre-processor) and learning only 2 learners enables someone to build 4 very successful multi-methods. Therefore, our recommendations to practitioners, who are willing to use multi-methods but lack the knowledge of machine learning algorithms are:

- Start with initial 2 learners and build the associated multi-methods
- See the performance of the current multi-methods
- Build new multi-methods only if you are not pleased with the performance of the current ones

6.6.2 Ensemble and Accuracy

Writing in the field of marketing, Armstrong reports in his 2007 study that the multi-method forecasts out-perform single-method forecasts [102]. He cites 31 studies where multiple source prediction consistently out-performs single source prediction by 3.4 to 23.4% (average = 12.5%). It is expected to see that a meaningful combination method, such as ours, results in a superior performance. On the other hand, it is impossible to cover all meaningful combination strategies in this single report. Our study adopts only one of the many proposed successful combination methods [103]. Work by Hogarth [104] demonstrates that a reduced inter-correlation between the prediction sources may be just as important as finding the prediction sources with highest expected accuracy (excluding the poor methods) when combining predictions." For this research, it means that understanding the inter-dependencies between different solo-methods could lead to clearer definitions of when to combine which particular solo-methods. In return, such a strategy could yield more robust multi-methods. For example a very promising future work would be to include the best neural network model into multi-methods.

6.7 Conclusions

The results of this chapter confirm the effect reported by Shepperd et al.: in Figure 6.2 and Figure 6.3 we see that a method's ranking can change by a certain amount δr. However, the results of this study are more optimistic. While some methods have very large δr, others do not. For the solo-methods shown in Figure 6.2, the better methods have smaller δr. An even better result is that when we combine these *superior* solo-methods, we elicit multi-methods that can attain almost zero δr. The benefits of using multi-methods that we observed as a result of the research reported in this chapter are:

- The multi-methods consistently out-perform most of the solo-methods;
- The performance of the multi-methods are more stable with the smallest ranking instability.
- As shown in Figure 6.5, the multi-methods avoid large errors that were seen in other methods.

Therefore, the 6^{th} principle to be reported in this thesis is:

Principle #6: Assemble Superior Methods

The contributions of the research following this principle are:

- A novel scheme for ensembling the best solo-methods.
- Stable multi-methods that outperform *all* solo-methods.

Chapter 7

Principle #7, Weighting Analogies is Over-elaboration

In this chapter, we will be investigating the analogy-based estimation (ABE from now on) methods. As we will see in the following paragraphs, there are thousands of ABE varieties that can be generated by altering different steps associated with ABE methods. In such a large space of possible options, it is important to investigate likely varieties. However, this investigation should be aiming two directions: 1) Introduction of new successful ABE variants; and 2) Identifying and pruning the less promising variants to reduce the large space of possibilities. In the next chapter of this thesis we will introduce the easy-path principle (see Chapter 8), which is a good example to the first direction of introducing new ABE variants. This chapter on the other hand is along the lines of the second direction, i.e. reducing the space of possible ABE variants.

Elsewhere [16, 67], we have studied a large number of different kinds of ABE methods in the literature, which have the common steps of:

- ABE generates estimates by sampling the neighborhood of some test instance.
- The exact sampling method is controlled by the *kernel method* which we divide into *uniform* and *non-uniform* (here after U-ABE and N-ABE, respectively).
- Uniform methods treat all items in the local neighborhood in the same way.
- Non-uniform methods weight those neighbors in different ways.

Our reading of the literature shows that different studies adopt different ways to handle:

- The selection of relevant features;
- The similarity function;
- The weighting method used in similarity function;
- The case subset selection method (a.k.a selected analogies or *k* value);
- And the adaptation strategy (a.k.a solution function)

Choice of different solutions to each of the above steps define a different ABE configuration. A detailed discussion on the studies using alternative configurations is given by Kocaguneli et al. [16]. In [16] it is shown that we can easily find over 17,000 different ways to configure ABE-style estimators. Then the question is: "How we select the right method from this large menagerie of possibilities?" One way is to try many options, then see what works best on the local data. Baker then Menzies et al., used exhaustive search (i.e. try all possible combinations) to find the best combinations of project features, learners and other variables [26, 61]. The CPU intensive nature of that approach begs the question: Is there a simpler way?

There is indirect evidence that there must be a simpler way. If we look at the size of the training data available for effort estimation, it is usually only a few dozen (or less) instances. For example:

- The data accessible to researchers in four recent publications [1,46,61,70] have median size of $13, 15, 33, 52$, respectively.
- The data sets used in this research (all public-data sets of Figure 2.1 except China) vary in size from 10 to 93 with a median of 28.

Given that the size of these data sets is so small, it seems reasonable to believe that (e.g.) complex multi-dimensional partitioning schemes (partition data into subgroups according to a criterion that uses multiple features/dimensions of the data and where each subgroup shares a common property, e.g. regression trees) reduce to more simplistic methods such as "take the median of the variables in the local region".

This is indeed the case. The experiments of this chapter show that, for ABE, the simplest kernel schemes are *most often as good as anything else*. This is an important result since it means that future researchers have less to explore (at least, in the field of ABE). Hopefully, if more researchers critically reviewed the space of options for their tools, then we will arrive at a much smaller and much more manageable sets of candidate effort estimation methods.

Below, we list some of the key concepts used in this research for convenience of the reader and provide brief definitions upfront. Detailed explanations and examples are provided in related sections.

- *Analogy:* The project instance from the training set, which will be used for estimating the effort for the test instance(s).
- *Kernel Density Estimation:* A non-parametric method for estimating a probability density function (PDF). In our case it acts like a PDF giving a probability value for the selected analogies one at a time.
- *Kernel:* A function that evaluates the difference (normalized by bandwidth) between the analogy (for which we want a probability value) and all the remaining training instances (see Equation 7.5).
- *Bandwidth:* A smoothing parameter telling the kernel how big of a neighborhood around the analogy in the training set is important.
- *Feature of a project:* One of the many variables defining a software project, e.g. lines of code (LOC), function points (FP) etc.
- *Instance Selection:* The process of selecting out project instances from the training set according to a distance function that are to be used in the estimation phase.
- *Feature Weighting:* Multiplying feature values with higher or lower values/weights to emphasize that they are more or less important, respectively.

To provide a focus this chapter, we will answer the following research questions:

RQ1 Is there evidence that non-uniform weighting improves the performance of ABE?

RQ2 What is the effect of different kernels for non-uniform weighting in ABE?

RQ3 What is the effect of different bandwidths?

RQ4 How do the characteristics of software effort datasets influence the performance of kernel weighting in N-ABE?

Note that, for reasons of space, our results will be presented in a summary format. For the full results, see http://goo.gl/qpQiD.

7.1 Motivation

This chapter specifically focuses on ABE for several reasons:

- It is a widely studied approach [13, 15, 28, 38, 46, 47, 70–74, 105, 106].
- It works even if the domain data is sparse [15].
- Unlike other predictors, it makes no assumptions about data distributions or an underlying model.
- When the local data does not support standard algorithmic/parametric models like COCOMO, ABE can still be applied.

Based on a literature review of just one sub-section of the field [67], we have found at least six dimensions that distinguish different ABE methods:

A : The distance measure used to compute similarity;

B : The "neighborhood" function that decides what is a "near" neighbor;

C : The method used to summarize the nearest neighbors;

D : The instance selection mechanism;

E : The feature weighting mechanism;

F : The method for handling numerics, e.g. logging, discretization, etc.

That review found in the literature three to nine variants of A, B, C, D, E, F (which combine to a total over 17,000 variants). Some of these variants can be ruled out, straight away. For example, for ABE that reasons only about the single nearest neighbor, all the summarization mechanisms return the same result. Also, not all the feature weighting techniques require discretization, thereby further decreasing the space of options. However, even after discarding some combinations, there are still thousands of possibilities to explore.

In our view, it is unacceptable that researchers continually extend effort estimation methods without trying to prune away the less useful variants. To that end, in previous work, we have tried to rank and prune estimation methods based on model selection [67] or feature weighting [26] or instance selection [16].

We have had much recent success in pruning different variants:

- For COCOMO-style data [20], only four variants were demonstrably better than the other 154 variants [67].
- Also, in non-COCOMO data, we have found 13 variants that perform better than 77 others [49].

This chapter is an exploration of kernel methods. Kernel methods are important since they comment on many of the options within A, B, C, D, E, F listed above:

- Simpler kernel methods mean simple neighborhood and summarization methods.
- In theory, better estimates could be generated by a smarter sampling of the neighborhood. For example, an intelligent selection of the kernel might compensate for data scarcity.

We study kernel estimation since, if the effort data corresponds to a particular distribution, then it would seem wise to bias that sampling by that distribution. Also, at least one other research team in the field of effort estimation have also begun exploring different kinds of kernel methods (e.g. inverse-ranked weighted mean) [1, 17]. The other kernel methods employed in our research are also explored by other researchers [59, 107–110]. For a detailed discussion on the effect of different commonly adopted kernel types, the reader can refer to Hardle et al. [110].

Despite the potential of kernel methods to improve effort estimation, it is an unexplored area. For the most part, researchers in this area propose kernel methods with minimal motivation or experimentation [1, 17, 111, 112]. Hence, prior to performing the experiments reported in this chapter, we believed that kernel methods would be a rich source of future insights into effort estimation:

- The space of sampling and weighting schemes seen in the software engineering (SE) literature is much smaller than that seen in other fields (see for example the literature from data mining or signal processing [59, 107, 108]).
- Hence, it seemed to us that a rigorous exploration of this under-explored area might be a worthy topic of research, perhaps applying methods not yet used in the SE literature.

The research explained in this chapter presents a rigorous exploration and leads to the the negative result summarized in the conclusion that simple kernel methods do as well as anything else, at least for ABE. Considering the characteristics of software effort datasets (small size, high levels of noise), it is wise to keep in mind that simple approaches may perform better than expected. Thereby, making the more complex alternatives a choice that one should approach with caution.

7.2 On the Value of Negative Results

While it would have been gratifying to have found a positive result (e.g. that some kernel method was very much better), it is important to report such negative results as well. A very thorough discussion on the value of negative results can be found in [113]. The fundamental question is whether a negative result poses a positive knowledge. Positive knowledge is defined by Browman et al. to be the ability of being certain, not being either right or wrong [113]. However, not all certain conclusions are *knowledge*. Common concerns are:

i. is the topic/hypothesis plausible,

ii. are the experiments sound,

iii. do the results propose *"negative evidence"* or *"non-conclusive search"* and

iv. will the reported results be valuable to future research.

As for ***i.***, research on weighting methods in ABE is quite plausible, see weighting method proposed earlier by Mendes et al. [1,17]. In that respect, our evidence of negative results serve the purpose of guiding research away from conclusions (such as kernel weighting can improve ABE performance) that would otherwise seem reasonable [113].

When presenting negative evidence it is crucially important to have *sound and extensive* experimentation (condition ***ii.***). This report rigorously investigates kernel weighting on *19* datasets subject to *3* performance measures through appropriate statistical tests.

The idea behind condition ***iii.*** is that *"one should disvalue inconclusive results"* [113], i.e. negative conclusions are more meaningful than *uncertainty*. The kernel weighting experiments of this chapter on a wide range of ABE variants are negative evidence to conclude that it does not improve ABE performance, thereby satisfying ***iii.*** Finally condition ***iv.*** questions the benefit of results to future research. After years of research, effort estimation still suffers from conclusion instability, i.e. proposed results are not widely applicable, they are bound to change w.r.t. different estimation methods and experimental conditions. Shepperd et al. list the likely causes leading to conclusion instability as the estimation models, performance measures, SEE datasets and sampling methods [4]. For more notes on conclusion instability see [49]. For stable conclusions, i.e. conclusions that are widely applicable w.r.t. causes of instability, retiring a considerable portion of search space is as important as the discovery of successful applications. The contribution of this work is through retirement of *2090* of ABE variants.

In this research we will compare the results of ABE0 framework (which was introduced in §2.7) with different non-uniform weighting strategies, i.e. with different N-ABE methods. Note that since ABE0 is a framework for U-ABE methods, in the rest of the chapter the two terms will be used interchangeably. N-ABE methods have been previously addressed in literature. For example inverse rank weighted mean (IRWM) was proposed by Mendes et al. [1]. IRWM method enables higher ranked analogies to have greater influence than the lower ones. Assuming that we have *3* analogies, the closest analogy (CA) gets a weight of $\frac{3}{\sum_{i=1}^{3} i}$, the second closest (SC) gets a weight of $\frac{2}{\sum_{i=1}^{3} i}$ and the last analogy (LA) gets $\frac{1}{\sum_{i=1}^{3} i}$.

7.3 Methodology

7.3.1 Kernel Density Estimation

Kernel density estimation (a.k.a. Parzen Windows) is a non-parametric technique used to estimate an unknown probability density function (PDF) [107, 114, 115]. Our short notes on kernel density estimation given here are based on the excellent tutorial by Duda et al. [115]. Therefore, reader is strongly suggested to see Chapter 4 of [115] for an in depth discussion.

The main idea behind non-parametric density estimation is rather simple, the density function can be viewed as the probability of seeing other samples from the same distribution in a given region. Think of the following intuitive example. Assume we have n points $(x_1, x_2, ..., x_n)$ that are independently and identically distributed (i.i.d) with respect to probability $p(x)$. Further assume that we define a region R with volume V. Obviously only a portion of the n points (say k-many) will fall into this region. We can use this fact to derive the following estimate for $p(x)$:

$$p(x) = \frac{k/n}{V} \tag{7.1}$$

According to this scenario, if we had 10 points (i.e. $n = 10$) and we had defined our sample volume to be a unit-cube that contained only 5 of these 10 points, our estimate for $p(x)$ would be $p(x) = \frac{5/10}{1} = 0.5$.

To see how this simple formula is used as the basis of kernel density estimation, temporarily assume that the region we sample from R is a hyper-cube centered at the origin with d dimensions. Given one edge-length of this hyper-cube is h, its volume becomes: h^d (the h value is also known as the bandwidth value). So as to find an expression for the number of points (i.e. k) within this region, we can define the following kernel function:

$$K(\rho) = \begin{cases} 1, & \text{if } |\rho| \leq 0.5 \\ 0, & \text{elsewhere} \end{cases} \tag{7.2}$$

Note that above kernel function (i.e. $K(\rho)$) is nothing but a unit hypercube centered at the origin. If we center this hyper-cube at x (a point for which we want to get the probability estimate), the number of samples falling within the hypercube (i.e. (k)) can be calculated as follows[1]:

$$k = \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{7.3}$$

Now if we replace the k value in Equation 7.1 with the expression of Equation 7.3, we get the estimate as:

$$p(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{V_n} K\left(\frac{x - x_i}{h}\right) \tag{7.4}$$

We should note that the kernel function (i.e. $K\left(\frac{x-x_i}{h}\right)$) is used for interpolation and each sample point in our space contributes to the estimate depending on its distance to the point x (for which we want to find the probability).

However, see that just using a hyper-cube function as the kernel function is rather limiting. To make $p(x)$ a more general and a proper PDF, we need to make sure that all the values it returns are greater than or equal to zero and it integrates to 1. This can be achieved by choosing the kernel function itself as a probability distribution function [116].

There are different kernel functions used to make the $p(x)$ a PDF [109]. This chapter explores the commonly used kernels, which are given in Figure 7.1 as well as IRWM [1,17]. IRWM is not actually proposed as a kernel method and it does not fully conform to the definition of standard kernel methods. However, due to the weighting strategy it proposes we can read it as an expert proposed kernel.

A literature review revealed that the selection of bandwidth (*h*) for kernels is more influential than the kernel types [109, 117]. Bandwidth *h* is fundamentally a scaling factor that controls how wide probability density function will spread, i.e. appropriate choice of *h* is critical to avoid under and over-smoothing [114,116]. To avoid both under and over-smoothing conditions we used various bandwidth values. One of the bandwidths we used is suggested by John et al., which is $h = 1/\sqrt{n}$ where h is the bandwidth and n is the size of dataset [108]. The other bandwidth values we used are: 2, 4, 8 and 16.

[1]Note that the effort values stored in software effort datasets are stored in a single column; hence our space is 1-dimensional. In other words, V_n in this formula will be just 1-dimensional too which is just the bandwidth value h, i.e. $V_n = h$.

Kernel Type	Formula
Uniform Kernel	$K(\rho) = \frac{1}{2}\mathbf{1}_{(\lvert\rho\rvert<1)}$
Triangular Kernel	$K(\rho) = (1 - \lvert\rho\rvert)\,\mathbf{1}_{(\lvert\rho\rvert<1)}$
Epanechnikov Kernel	$K(\rho) = \frac{3}{4}(1 - \rho^2)\,\mathbf{1}_{(\lvert\rho\rvert<1)}$
Gaussian Kernel	$K(\rho) = \frac{1}{\sqrt{2\pi}}e^{(\frac{-1}{2}\rho^2)}$
IRWM Kernel	—

Figure 7.1: The formulas for different kernels used in this study, where $\mathbf{1}_{(|x|<1)}$ is the indicator function. In formulas $\rho = \frac{x-X_i}{h}$. Note that IRWM kernel has different characteristics that standard kernels.

7.3.2 Weighting Method

Assume that our dataset of size n is divided into two sets:

- $A = \{x_1, ..., x_i, ..., x_k\}$ (effort values of the selected ***A***nologies with cardinality k and x_i ($i \in \{1...k\}$) representing an element of **A**)
- and $R = \{x_1, ..., x_j..., x_{n-k}\}$ (effort values of the ***R***est of the dataset with cardinality $n - k$ and x_j ($j \in \{1...(n - k)\}$) representing an element of **R**).

We build the kernel density estimation on R and evaluate the resulting function at instances of A. Equation 7.5 shows the probability calculation with kernel density estimation. In Equation 7.5 the kernel K is built on training data $x_j \in R$ and is evaluated at analogy x_i for a bandwidth of h. After scaling these probability values to 0-1 interval according to Equation 7.6, we use them as weights for analogies. After calculating $weight_{x_i}$ for each analogy, we update their actual effort values according to Equation 7.7.

$$f(x_i, h) = \frac{1}{nh} \sum_{x_j \in R} K\left(\frac{x_i - x_j}{h}\right) \tag{7.5}$$

$$weight_{x_i} = \frac{f(x_i, h) - max(f(x_i, h)values)}{max(f(x_i, h)values) - min(f(x_i, h)values)} \tag{7.6}$$

$$updatedEffort_{x_i} = actualEffort_{x_i} * weight_{x_i} \tag{7.7}$$

Uniform vs. Non-Uniform Weighting

The fundamental difference between N-ABE and U-ABE methods is that in U-ABE analogies are given uniform weights and their actual effort values are used in an *as is* manner, whereas in N-ABE analogies are assigned different weights and their actual effort values are multiplied by these weight values. As for U-ABE, we defined a base method that we call ABE0 and for N-ABE we use 5 different kernel methods.

One point that needs further clarification is the use of uniform kernel as a N-ABE method. Figure 7.2 succinctly illustrates the difference between uniform kernel being a N-ABE method and ABE0 being a U-ABE method. ABE0 assumes equal importance of *all* instances and assigns equal probabilities. A uniform kernel would assign equal non-zero probabilities to *only a certain portion* of the instances, whereas the rest of the instances would be assigned a weight of zero (i.e. they would be ignored).

7.3.3 Experiments

Our experimental settings aim at comparing the performance of standard U-ABE (ABE0) to that of N-ABE. To separate train and test sets we use leave-one-out method, which entails selecting 1 instance out of a dataset of size n as the test set and using the remaining $n - 1$ instances as the training set. For each test instance, we run ABE0 and N-ABE separately and store their estimates. As the analogy number is reported to play a critical role in estimation accuracy [47], both for U-ABE and N-ABE methods, we tried different *k* values.

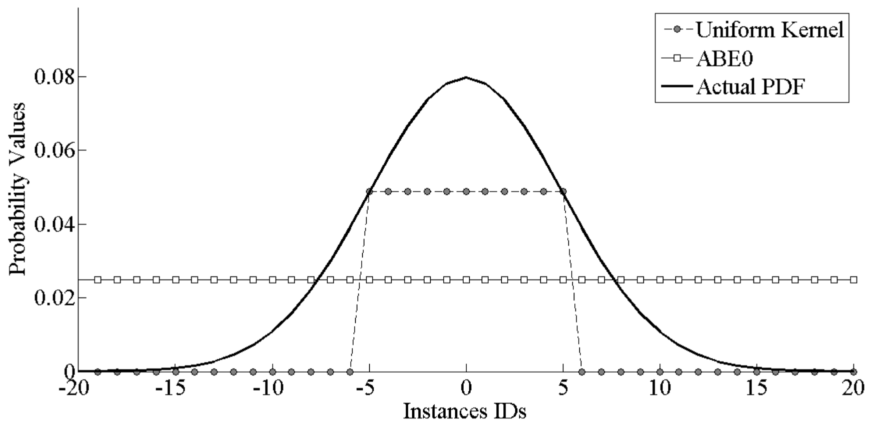


Figure 7.2: In the case of ABE0 all instances are given equal probability values, hence equal weights. However, uniform kernel prefers some instances over the others: Only a certain portion of the instances are given equal non-zero weights.

We use 2 forms of ABE methods (uniform and non-uniform weighting) induced on 19 datasets for 5 different k values. The k values we used in our research are: $k \in \{3, 5, 7, 9, best\}$. $best$ is a pseudo-best k value that is selected for each individual test instance through a process, in which we randomly pick up 10 instances from the training set and select the lowest error yielding k value as the $best$. Since pseudo-best k includes a random procedure, to hinder any particular bias that would come from the settings of a single experiment, we repeated the afore mentioned experimental procedure 20 times. Note that $k > 1$ for $\forall k$, because for $k = 1$ U-ABE and N-ABE would be equivalent. In addition, we use 5 different kernels (Uniform, Triangular, Epanechnikov, Gaussian and IRWM) with 5 bandwidth values in N-ABE experiments. To further explore field of SEE, we investigate a total of 2090 different settings in this research:

- U-ABE Experiments: 95 settings
  - *19 datasets * 5 k values = 95*
- N-ABE Experiments: 1995 settings
  - Standard Kernels: *19 datasets * 5 k values * 4 kernels * 5 bandwidths = 1900*
  - IRWM: *19 datasets * 5 k values = 95*

7.4 Results

To see the effect of kernel weighting, we studied 19 datasets and 3 different performance measures. For each performance measure we tried 4 different kernels subject to 5 different bandwidths, plus the IRWM kernel (which does not have a bandwidth concept) and reported associated $win, tie, loss$ statistics.

Figure 7.3 shows a sample of our results. It reports the $win, tie, loss$ statistics of Desharnais dataset for ABE0 and N-ABE through Gaussian kernel. For each dataset we have 4 such tables (one for each kernel), so for all datasets there are *19 Datasets* $\times$ *4 tables = 76 tables*. For the IRWM kernel there will be another *19 datasets* $\times$ *1 kernel = 19 tables*. Hence, in total, our results comprise *76 + 19 = 95 tables* to report. All these tables are available on line at http://goo.gl/qpQiD (user-name: guest, password: guest). However, for space reasons, we summarize those tables as follows.

In Figure 7.3 we see that each row reports $win, tie, loss$ statistics of ABE0 methods (*k=3,5,7,9,best*) as well as N-ABE methods (*k=[3,5,7,9,best]+kern* where *kern* stands for kernel weighting) subject to a particular performance measure. Similarly, each column shows the $win, tie, loss$ statistics associated with a particular bandwidth value. As can be seen in Figure 7.3, for Desharnais dataset ABE0 methods always have higher win values and always have a $loss$ value of 0, meaning that they never lose against N-ABE methods. That is, by summarizing each row/column intersection we can see that the performance of ABE0 is never improved by N-ABE.

Figure 7.4 and Figure 7.5 repeat that summarization process for all 19 datasets and all kernel/bandwidth combination:

- Each row of these summary figures shows the comparison of ABE0 performance to that of N-ABE subject to 3 different performance measures.
- Every kernel/bandwidth intersection in Figure 7.4 and Figure 7.5 has 3 symbols corresponding to MdMRE, MAR and Pred(25) comparisons from left to right.
- Each of these 3 symbols can have 3 values: $-, +, o$.
  - "$-$" means that N-ABE decreased the accuracy of ABE0;

		h=1/sqrt(size)			h=2			h=4			h=8			h=16		
		WIN	TIE	LOSS	WIN	TIE	LOSS	WIN	TIE	LOSS	WIN	TIE	LOSS	WIN	TIE	LOSS
MdMRE	*k*=3	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=5	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=7	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=9	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=best	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=3+kern	80	0	100	80	0	100	80	0	100	80	0	100	80	0	100
	k=5+kern	0	60	120	60	0	120	60	0	120	60	0	120	60	0	120
	k=7+kern	0	60	120	20	20	140	20	20	140	20	20	140	20	20	140
	k=9+kern	0	60	120	0	0	180	0	0	180	0	0	180	0	0	180
	k=best+kern	0	60	120	20	20	140	20	20	140	20	20	140	20	20	140
MAR	*k*=3	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=5	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=7	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=9	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=best	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=3+kern	0	80	100	60	20	100	60	20	100	60	20	100	60	20	100
	k=5+kern	0	80	100	0	80	100	0	80	100	0	80	100	0	80	100
	k=7+kern	0	80	100	0	60	120	0	60	120	0	60	120	0	60	120
	k=9+kern	0	80	100	0	60	120	0	60	120	0	60	120	0	60	120
	k=best+kern	0	80	100	0	60	120	0	60	120	0	60	120	0	60	120
Pred(25)	*k*=3	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=5	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=7	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=9	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=best	100	80	0	100	80	0	100	80	0	100	80	0	100	80	0
	k=3+kern	60	0	120	80	0	100	80	0	100	80	0	100	80	0	100
	k=5+kern	20	60	100	60	0	120	60	0	120	60	0	120	60	0	120
	k=7+kern	0	60	120	0	20	160	0	20	160	0	20	160	0	20	160
	k=9+kern	0	60	120	20	0	160	20	0	160	20	0	160	20	0	160
	k=best+kern	0	60	120	20	20	140	20	20	140	20	20	140	20	20	140

Figure 7.3: Desharnais dataset $win, tie, loss$ statistics for ABE0 and N-ABE through Gaussian kernel. For each dataset we have 4 of these tables (one for each kernel). In total it amounts to *19 Datasets* $\times$ *4 tables = 76 tables*. In addition we have another *19 datasets* $\times$ *1 kernel = 19 tables* from IRWM kernel. It is infeasible to include all the tables into this chapter, therefore an executive summary of $76 + 19 = 95$ tables is provided in Figure 7.4. Furthermore, we provide all 95 tables in excel format at http://goo.gl/qpQiD.

Dataset	Kernel	h=1/sqrt(size)	h = 2	h = 4	h = 8	h = 16
Coc81	**Uniform**	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
	Triangular	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
	Epanechnikov	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
	Gaussian	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
Coc81e	**Uniform**	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
	Triangular	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
	Epanechnikov	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
	Gaussian	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
Coc81o	**Uniform**	$-o-$	$-oo$	$-oo$	$-oo$	$-oo$
	Triangular	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
	Epanechnikov	$---$	*ooo*	*ooo*	*ooo*	*ooo*
	Gaussian	$-o-$	*ooo*	*ooo*	*ooo*	*ooo*
Coc81s	**Uniform**	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
	Triangular	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
	Epanechnikov	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
	Gaussian	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
Ns93	**Uniform**	$-o-$	$-o-$	$-o-$	$-o-$	$-o-$
	Triangular	$-o-$	$-o-$	$-o-$	$-o-$	$-o-$
	Epanechnikov	$-o-$	*ooo*	*ooo*	*ooo*	*ooo*
	Gaussian	$-o-$	*ooo*	*ooo*	*ooo*	*ooo*
Ns93c1	**Uniform**	$---$	$---$	$---$	$---$	$---$
	Triangular	$---$	$-o-$	$-o-$	$-o-$	$-o-$
	Epanechnikov	$---$	$-o-$	$-o-$	$-o-$	$-o-$
	Gaussian	$-o-$	$-o-$	$-o-$	$-o-$	$-o-$
Ns93c2	**Uniform**	*ooo*	$-o-$	$-o-$	$-o-$	$-o-$
	Triangular	$-o-$	*ooo*	*ooo*	*ooo*	*ooo*
	Epanechnikov	$-o-$	*ooo*	*ooo*	*ooo*	*ooo*
	Gaussian	*ooo*	*ooo*	*ooo*	*ooo*	*ooo*
Ns93c5	**Uniform**	$---$	$-o-$	$-o-$	$-o-$	$-o-$
	Triangular	$---$	*ooo*	*ooo*	*ooo*	*ooo*
	Epanechnikov	$---$	*ooo*	*ooo*	*ooo*	*ooo*
	Gaussian	$---$	*ooo*	*ooo*	*ooo*	*ooo*

Figure 7.4: Nine data sets comparing ABE0 to N-ABE. For every row in each cell, there are three symbols indicating the effect of N-ABE w.r.t. 3 different error measures. From left to right, the first symbol stands for N-ABE effect w.r.t. MdMRE, the second symbol w.r.t. MAR and the third one w.r.t. Pred(25). A "+" indicates that for majority of k values (at least 3 out of 5 k values), N-ABE improved ABE0 in terms of $win - loss$ values. "$-$" indicates that N-ABE decreased the performance of ABE0 in the majority case. If the former conditions do not satisfy, then a "o" symbol is assigned. Note that the dataset come from Figure 2.1, yet the dataset names are abbreviated to 3 to 5 letters due to space constraints.

Dataset	**Kernel**	**h=1/sqrt(size)**	**h = 2**	**h = 4**	**h = 8**	**h = 16**
Des	**Uniform**	− − −	− − −	− − −	− − −	− − −
	Triangular	*ooo*	− − −	− − −	− − −	− − −
	Epanechnikov	− − −	− − −	− − −	− − −	− − −
	Gaussian	− − −	− − −	− − −	− − −	− − −
DesL1	**Uniform**	− − −	− − −	− − −	− − −	− − −
	Triangular	*ooo*	−*o*−	−*o*−	−*o*−	−*o*−
	Epanechnikov	− − −	−*o*−	−*o*−	−*o*−	−*o*−
	Gaussian	− − −	−*o*−	−*o*−	−*o*−	−*o*−
DesL2	**Uniform**	− − −	− − −	− − −	− − −	− − −
	Triangular	*ooo*	− − −	− − −	− − −	− − −
	Epanechnikov	− − −	−*o*−	−*o*−	−*o*−	−*o*−
	Gaussian	− − −	−*o*−	−*o*−	−*o*−	−*o*−
DesL3	**Uniform**	−*o*−	−*o*−	−*o*−	−*o*−	−*o*−
	Triangular	−*o*−	*ooo*	*ooo*	*ooo*	*ooo*
	Epanechnikov	−*o*−	*ooo*	*ooo*	*ooo*	*ooo*
	Gaussian	−*o*−	*ooo*	*ooo*	*ooo*	*ooo*
SDR	**Uniform**	−*o*−	−*o*−	−*o*−	−*o*−	−*o*−
	Triangular	+ + −	+*o*−	+*o*−	+*o*−	+*o*−
	Epanechnikov	− + −	+ + −	+ + −	+ + −	+ + −
	Gaussian					
Albr	**Uniform**	− − −	− − −	− − −	− − −	− − −
	Triangular	− − −	−*o*−	−*o*−	−*o*−	−*o*−
	Epanechnikov	− − −	−*o*−	−*o*−	−*o*−	−*o*−
	Gaussian	− − −	−*o*−	−*o*−	−*o*−	−*o*−
Finn	**Uniform**	− − −	− − −	− − −	− − −	− − −
	Triangular	*ooo*	− − −	− − −	− − −	− − −
	Epanechnikov	− − −	−*o*−	−*o*−	−*o*−	−*o*−
	Gaussian	− − −	−*o*−	−*o*−	−*o*−	−*o*−
Kem	**Uniform**	−*o*−	−*o*−	−*o*−	−*o*−	−*o*−
	Triangular	− − −	*ooo*	*ooo*	*ooo*	*ooo*
	Epanechnikov	− − −	*ooo*	*ooo*	*ooo*	*ooo*
	Gaussian	− − −	*ooo*	*ooo*	*ooo*	*ooo*
Maxw	**Uniform**	− − −	− − −	− − −	− − −	− − −
	Triangular	− − −	*ooo*	*ooo*	*ooo*	*ooo*
	Epanechnikov	− − −	*ooo*	*ooo*	*ooo*	*ooo*
	Gaussian	− − −	*ooo*	*ooo*	*ooo*	*ooo*
Miy94	**Uniform**	− − −	− − −	− − −	− − −	− − −
	Triangular	− − −	− − −	− − −	− − −	− − −
	Epanechnikov	− − −	−*o*−	−*o*−	−*o*−	−*o*−
	Gaussian	− − −	−*o*−	−*o*−	−*o*−	−*o*−
Tel	**Uniform**	− − −	− − −	− − −	− − −	− − −
	Triangular	−*o*−	−*o*−	−*o*−	−*o*−	−*o*−
	Epanechnikov	− − −	−*o*−	−*o*−	−*o*−	−*o*−
	Gaussian	− − −	−*o*−	−*o*−	−*o*−	−*o*−

Figure 7.5: Ten more data sets comparing ABE0 to N-ABE. Same format as Figure 7.4.

  - "o" means ABE0 and N-ABE are statistically same;
  - "$+$" shows that ABE0 accuracy was improved through kernel weighting (i.e. N-ABE has a better performance than ABE0).

We assign the symbols "+" or "$-$" if the performance associated with the majority of the k-values (at least 3 out of 5) are improved or degraded by N-ABE (in terms of $win - loss$). If there is no change, we assign a "o" symbol to that setting.

Observe that in all these summaries:

- There is only one dataset (SDR in Figure 7.5) where N-ABE provides a performance improvement in certain cases. Even for that dataset there are 15 "+" symbols and 21 "$-$" symbols, meaning that most of the time N-ABE is still destructive.
- In 18 other datasets, which is $\frac{18}{19} = 95\%$ of all the datasets, there is not a single case where N-ABE improves the performance of ABE0.

Note that these summary tables contain results from different performance criteria (MdMRE, MAR, Pred(25)) as well as kernels and bandwidths. Therefore, our conclusion from Figure 7.4 and Figure 7.5 is that *"non-uniform weighting through standard kernel methods does not improve the performance of ABE"* holds in the majority case across different datasets and error measures.

Another summary table is given in Figure 7.6. Figure 7.6 is very similar to Figure 7.4 in the sense that it summarizes the performance of N-ABE over 19 datasets w.r.t. three different performance measures. The difference is that Figure 7.4 summarizes the results of standard kernel methods, whereas in Figure 7.6 we see the N-ABE performance under an *expert-based* kernel, i.e. IRWM. Although there are important differences between standard and expert-based kernels (IRWM has no bandwidth parameter), the results seen in Figure 7.6 is quite similar to those of Figure 7.4. As can be seen in Figure 7.6, there is not a single case where N-ABE (under IRWM kernel) improves the performance of ABE0. Furthermore, the amount of "$-$" symbols is much more than "o", meaning that N-ABE decreases the performance of ABE0 most of the time.

Dataset	**Improvement**		
	MdMRE	**MAR**	**Pred(25)**
Cocomo81	−	*o*	−
Cocomo8e	*o*	*o*	*o*
Cocomo8o	−	−	−
Cocomo8s	*o*	*o*	*o*
Nasa93	−	−	−
Nasa93_center_1	−	−	−
Nasa93_center_2	−	*o*	−
Nasa93_center_5	−	−	−
Desharnais	−	−	−
DesharnaisL1	−	−	−
DesharnaisL2	−	−	−
DesharnaisL3	−	*o*	−
SDR	−	*o*	−
Albrecht	−	−	−
Finnish	−	−	−
Kemerer	−	−	−
Maxwell	−	−	−
Miyazaki94	−	−	−
Telecom	−	−	−

Figure 7.6: The comparison of ABE0 to N-ABE under IRWM kernel. Similar to Figure 7.4 three symbols indicate the effect of N-ABE w.r.t. 3 different error measures and "+" indicates that for majority of k values N-ABE improved ABE0 in terms of $win - loss$ values. A "−" symbol indicates a decrease and a "o" symbol indicates neither decrease nor increase. Notice that subject to IRWM kernel, N-ABE fails to improve ABE0 w.r.t. 3 different performance measures.

7.5 Conclusions

RQ1. Is there evidence that non-uniform weighting improves the performance of ABE? The results of our experiments do not show such an evidence. On the contrary, for almost all the settings ABE0 yields much better results than N-ABE methods.

RQ2. What is the effect of different kernels for non-uniform weighting in ABE? There are only slight variations in performance when different kernels are used. However, these variations do not follow a definite pattern and they are far from being considerable.

RQ3. What is the effect of different bandwidths? Change of bandwidths shows a random and insignificant effect, which is very similar to that of kernel change effect. Therefore, we cannot say that applying different bandwidths has a certain effect on N-ABE performance.

RQ4. How do the characteristics of software effort datasets influence the performance of kernel weighting in N-ABE? Effort datasets are much smaller than most of the datasets in different domains. The dependent variable (effort value of a completed project) is highly variable. Furthermore, the attribute values are very open to personal judgment and error. All these factors suggest that non-parametric methods may be failing due to inherent characteristics of software effort data.

Following the results presented in this chapter, we define the 7^{th} principle as follows:

Principle #7: Weighting Analogies is Over-elaboration

Note that the results presented in this chapter are negative in characteristic and we see the contributions of this chapter as follows:

- Investigation of an unexplored and promising ABE option of kernel-weighting.
- An extensive experimentation of 2090 ABE scenarios, which reduces the ABE variants space to be explored.

Chapter 8

Principle #8, Use Easy-path Design

The principle that will be recommended at the end of this chapter is the use of a design method called easy-path, which fundamentally recommends the removal of the training cases that confuse the estimation method. Note that the term "confuse" refers to the removal of the training instances for which the estimation method yields high error rates. The research presented in this chapter uses the easy-path design to augment an ABE method. However, the principle of easy-path can also be applied to different contexts. For example, in another study we used easy-path design to configure a greedy agglomerative clustering-based estimation method [16]. Since this principle worked for different data sets and different contexts of SEE, we use it as one of the principles of SEE. The rest of the chapter explains the details of using easy-path in the context of ABE.

The standard ABE approach is to *fix* one parameter for each design alternative and try to find the best parameter settings. Kocaguneli et al. report that (see Section 2 of [16]) different ABE parameter settings seen in the SEE literature exceed $17,000$ configurations. It is reported that any *static-parameter* ABE method is suboptimal [24], and trying out tens of thousands alternatives is impractical.

Although there is a bewildering number of different options to set up an ABE method, by following the related SEE literature we are able to design a baseline method, more commonly known as ABE0, which is introduced in §2.7.

The fundamental design option in ABE0 is the number of analogous projects from the historical dataset (the *k* value) to be used. In his 2008 study, Keung reports that any *static k* value to be used in ABE0 settings would result in a suboptimal performance [24]. He also proposes a theoretical maximum prediction accuracy (TMPA) as the skyline (the highest performance) for ABE0 methods. That is, by using leave-one-out cross validation (a.k.a. Jackknifing) it has been observed that different test instances are best estimated by different *k* values (dynamic *k*). Then the question is how to best explore the space between the skyline and the suboptimal *static-k* approaches?

One alternative is to apply brute-force techniques and try out as many settings as possible. This is an undesirable option due to intensive computations and run-times, which makes it impractical to be used in software organizations. Furthermore, brute-force techniques are non-adaptive and inflexible. For changing situations due to addition/deletion of new projects, they would require a complete re-calibration.

Another alternative may be to use smart, low-cost heuristics that can perform comparable or -if possible- better than *static* approaches. One such heuristic called *"easy path principle"* is proposed by Kocaguneli et al. [16]. The heuristic is based on the following simple principle:

> *Find the situations that "confuse" the estimation and remove those situations prior to estimation.*

The application of *easy-path* principle on a variance-based ABE0 variant proved most beneficial (for detailed results see [16]). In [16], an ABE0 variant -so called TEAK- is proposed as a heuristic alternative to brute-force approaches. This heuristic is based on the assumption that *"locality implies homogeneity"*: Spatially close instances (locality) should have similar effort values (homogeneity), i.e. the variance in effort values between close instances should be relatively small. Therefore, *"high-variance"* regions in the dataset violate the assumption on which the learner is built and hence *"confuse"* the estimation. TEAK works by removing the regions of *"high-variance"* in a cluster tree of training instances.

We have observed that there is more to incorporate from a typical CBR-cycle (see Figure 2.6) into ABE [51, 52]. For example, the *"retain"* phase in CBR dictates only keeping the most useful and relevant instances of the past experience (the training instances) in the dataset. Following the prior work on SEE [16,24] and CBR [51,52], we propose a new heuristic based on the same design principle of *"easy-path"* but on a different assumption:

> *"TMPA indicates the best-set to be retained."*.

The details of the proposed method is in §8.2.3.

8.1 Motivation

Standard ABE methods use static-*k* values (i.e. static number of analogies) in the estimation. However, static-*k* approaches are sub-optimal, i.e. best estimates for different projects are attained with different *k*-values. Methods that would *dynamically* choose different number of analogies for each test instance are important for the performance improvement of ABE. Keung has proposed a dynamic-*k* approach to indicate the theoretical maximum prediction accuracy (TMPA) of standard ABE methods [24]. As noted in [24] there is a considerable space between static-*k* and TMPA for performance improvement.

TMPA is proposed as a conceptual method to show the possible space of improvement. However, it is not applicable in an actual estimation setting: Since we do not know the effort value of a test instance, we cannot know the best dynamic-*k* for that test instance. In this chapter we will address this practicality issue and propose a Dynamic-ABE method (from now on called as "D-ABE"). As it is shown in Section §11.6, the proposed D-ABE method significantly improves the performance of static-*k* ABE methods. Furthermore, it is able to cover an important portion of the space available (up to 32%) for performance improvement between static-*k* and dynamic-*k* methods.

8.2 Designing a Dynamic ABE0 Variant

8.2.1 Easy-path Principle

Easy-path is a design option for implementing suboptimal methods, i.e. it is a *heuristic* rather than standard practice. The standard practice for maturing prediction systems is through adding in new mechanisms to handle hard cases (cases for whom estimation accuracy is lower). AdaBoost [118] is a fine example for this general case: In a system of consecutive learners, each learner focuses on the cases, for which the prior learner was unsuccessful.

With the easy path principle we adopt a different strategy, which entails removing the hard-cases in the training set that would confuse the estimator. In fact focusing on just the easy cases (cases for whom estimation accuracy is higher) could have been problematic. Only exploring the easy cases might have meant that the estimator will perform poorly on the hard test cases.

However, previously we observed that this is not the case: Easy-path design strategy has proven to be very successful for a variance-based ABE0 variant [16]. Having positive results from such a design principle means that we are able to find short-cuts that simplifies effort estimation and avoids brute-force approaches.

8.2.2 TMPA: The skyline for ABE0

Following the success of easy-path in designing a prior ABE0 variant, we use the same design methodology to propose another *dynamic-k* ABE0 variant. The new method mimics the behaviour of an *"optimum"* ABE0 method called *theoretical maximum prediction accuracy* (TMPA), which is proposed by Keung et al. [24]. TMPA is a *"dynamic"* ABE0 method. It selects a test instance according to LOO cross-validation and tries out every possible *k* value. TMPA works as follows:

- Start executing on a dataset of size N
- Use leave-one-out cross validation to pick test instances one at a time
- Try all the *k* values from 1 to $N - 1$ and store the MRE values
- Pick the lowest MRE yielding *k* and return the estimate

As the name suggests, TMPA is a theoretical method used to define the highest prediction accuracy for ABE0. Since we cannot know the lowest MRE yielding *k* for an actual test instance (as the independent variable's value -effort- is kept hidden from the learner), TMPA is defined as a performance skyline for ABE0 rather than an actual learner.

The initially proposed TMPA [24] tries out all possible *k* values from 1 to $N - 1$. However, in this research we will restrict our attention on the *k* values from 1 to 5. The reasons for this focus are:

- The frequency distribution of the best-*k* values given by Keung [24] shows that $[1..5]$ interval has the highest frequency.
- An exhaustive search of all the *k* values would defeat the initial purpose of using a heuristic.
- The use of low *k* values is consistent with prior results. For example, Li et al. [46] suggest that a standard method is to always use $1 \leq k \leq 5$ nearest projects. Also:

- $k = 1$ is used by Lipowezky et al. [119] and Walkerden & Jeffery [15];
- $k = 2$ is used by Kirsopp & Shepperd [120]
- $k = 1, 2, 3$ is used by Mendes el al. [1]

8.2.3 Using Easy-path for a Dynamic ABE Method

Similar to TMPA, the proposed new method uses a dynamic number of *k* values for each test instance, hence it is called *"Dynamic-ABE"* (D-ABE). The design of D-ABE is carried out through the easy-path in 5 steps.

Stap 1. Select a prediction system: We select TMPA [24] as the base for the easy-path design. TMPA is a *theoretical* learner used to define the limits of ABE0 methods using *static-k* values. Our motivation in using TMPA is to implement a *practical* method that is *significantly* better than ABE0 methods using *static-k* values. In other words, we make use of the assumptions of TMPA to come up with a *dynamic* method to explore the performance space left between TMPA and *static* ABE0 variants.

Step 2. Identify the predictor's essential assumption(s): TMPA assumes that not all the training instances are candidates to be *retained*. By choosing the best performance-yielding *k-value* for each training instance, TMPA defines a best and a worst-set of training examples to be and not to be retained, respectively. The best/worst-set idea is used to pinpoint training examples that give high and low performance. We mark low-performance training instances *"not to be retained"* for the estimation of the test instance. The assumption in this step is:

TMPA indicates the best-set to be retained.

Step 3. Recognize when those assumption(s) are violated: Run TMPA on the training set via LOO and get the best estimate given by TMPA for each training instance. Then sort the training instances according to an error measure starting from the lowest error to the highest error. The training instances with error values close to the highest error are said to *violate* the assumption, because even with the best instances retained by TMPA, the performance of those instances is still poor. We used magnitude of relative error (MRE) as the error measure to define the best/worst estimate.

Step 4. Remove assumption-violating cases: After sorting the training instances (in Step 3) from the lowest MRE to the highest MRE, in this step we chop-off the training instances that have MRE values within α percent ($\alpha = 60\%$ in our experiments) of the worst MRE. This filter is used to define and chop-off the assumption violating training instances, i.e. it is used to define which training instances are to be *retained*. We store the retained instances together with their best estimates as yielded by TMPA in Step 3.

Step 5. Execute the modified prediction system: Upon arrival of a test intance, we execute ABE0 on the retained training set of Step 4. The closest retained training instance to the test instance is *retrieved*. The best effort estimate (given by TMPA and stored in the previous step) for the retrieved neighbor is returned as the estimate for the test instance. Although D-ABE uses the estimate of the closest neighbor (i.e. *k=1*) in the retained training instances, it still implicitly uses a *dynamic-k* value. Because note that the *k* values used for the estimates of the training instances were dynamically selected by TMPA in Step 3.

8.2.4 Noise Removal in CBR

Chopping-off training instances on the basis of TMPA performance and retaining only the ones with low error rates can be defined as a performance-based noise removal. In fact noise removal in CBR systems is quite common. This section reviews the studies that use noise removal in the context of CBR and explain why noise removal is an appropriate complement to a CBR.

Segata et al. note that, unlike other machine learning approaches (e.g. post-pruning in decision trees) CBR techniques lack a noise resistant mechanism [121]. Particularly for CBR methods applied on SEE datasets, the lack of such a mechanism is a huge drawback. Indeed various studies have shown that complementing CBR methods with a noise-removal mechanism significantly improves their performance on SEE datasets [16, 19, 46, 61, 122–124]. It has been shown that when complemented with a noise removal mechanism, CBR methods attain significantly better results [22, 23].

Segata et al. [121] follow the taxonomies proposed by Wilson et al. [125] and Brighton et al. [126] to define a taxonomy of noise removal techniques w.r.t. their purpose. Taxonomy of Segata et al. divides noise removal techniques into two categories: 1) Competence preservation and 2) competence enhancement techniques. The first group of noise removal techniques (competence preservation) aim at reducing the size of the instance-base for the purposes of low memory requirement, while preserving the performance. The second group aims at performance improvement when reducing the instance-base. Since SEE datasets are already relatively small, memory concerns are insignificant for us. Our approach (D-ABE) aims at performance improvement by removing noisy (low-performing) training instances. Hence, according to this taxonomy D-ABE appears to be a competence enhancement noise removal technique.

8.3 Experimental Rig

In order to evaluate the performance of a method, the ideal case would be to learn from past data and evaluate the model on *new*, *unseen* data. However, this strategy would be too impractical for experimentation: Imagine waiting for months (even years) for new software projects to complete so that you can finally test your new method. Cross-validation strategies simulate this ideal (yet impractical) scenario on historical datasets of completed projects.

There is a wide range of cross-validation strategies in SEE, unfortunately there is no consensus on which sampling method to use in SEE studies. We address the issue of which sampling method to use in the last principle of this thesis (see Chapter 12), where different sampling methods are empirically compared according to concerns of statistical properties (bias and variance). The recommendation of Chapter 12 is to use LOO in SEE studies. Hence, we also adopt LOO in the experimentation of the current chapter.

8.4 Results

8.4.1 Identify Datasets to be Explored by D-ABE

D-ABE is a sub-optimal heuristic whose aim is to improve ABE0 methods that use a static-*k* value. However, not all the datasets are feasible for further ABE0 performance improvement, i.e. for some datasets (so called *shallow* datasets) even TMPA is incapable of any improvement over static-*k* methods. For the shallow datasets, where even an optimum strategy -TMPA- is unable to provide an improvement, it is futile to search for an improvement via a sub-optimal heuristic.

Datasets	MMRE			MdMRE			Pred(25)		
cocomo81	5	0	0	5	0	0	5	0	0
cocomo81e	5	0	0	5	0	0	5	0	0
cocomo81o	2	3	0	2	3	0	2	3	0
cocomo81s	4	1	0	4	1	0	4	1	0
desharnais77	5	0	0	5	0	0	5	0	0
desharnaisL1	5	0	0	5	0	0	5	0	0
desharnaisL2	5	0	0	5	0	0	5	0	0
desharnaisL3	1	4	0	1	4	0	1	4	0
nasa93	5	0	0	5	0	0	5	0	0
nasa93_center_1	0	5	0	0	5	0	0	5	0
nasa93_center_2	4	1	0	4	1	0	4	1	0
nasa93_center_5	5	0	0	5	0	0	5	0	0
sdr	1	4	0	1	4	0	1	4	0
albrecht	3	2	0	3	2	0	3	2	0
finnish	5	0	0	5	0	0	5	0	0
kemerer	4	1	0	4	1	0	4	1	0
maxwell	5	0	0	5	0	0	5	0	0
miyazaki94	5	0	0	5	0	0	5	0	0
telecom1	2	3	0	2	3	0	2	3	0
TOTAL	71	24	0	71	24	0	71	24	0

Figure 8.1: The win (**w**), tie (**t**) and $loss$ (**l**) values of TMPA compared to ABE0 with $k \in \{1, 2, 3, 4, 5\}$. The *shallow* datasets where TMPA does not win against most of the ABE0 methods (**w**< 3) are highlighted. In these cases TMPA is unable to improve the static-*k* methods. Shallow datasets will be excluded in the analysis as there is not enough space for further improvement of static-*k* methods.

Therefore, we first identified the shallow datasets (highlighted rows of Figure 8.1) and excluded them from evaluation. As for the exclusion, we expect TMPA to have a win value of 2 or less, i.e. we expect ABE0 methods to tie with TMPA in the majority case (3 or more out of 5 comparisons). Note that since TMPA is compared against 5 static *k*-values ($k \in \{1, 2, 3, 4, 5\}$), TMPA can have a maximum a win value of 5.

Once we remove the shallow datasets of Figure 8.1, we get the datasets targeted for improvement. Figure 8.2 shows **w/t/l** values of TMPA *only* for the *target datasets*. In Figure 8.2 TMPA is compared to 5 static *k*-values ($k \in \{1, 2, 3, 4, 5\}$); hence, TMPA can have a maximum win value of 5. Since we have 14 target datasets for improvement, the *TOTAL* number of **w/t/l** for each error measure (last row of Figure 8.2) is *14 datasets* $\times$ *5 = 70*.

We see in Figure 8.2 (see the last row) that -as expected- TMPA never loses against any ABE0 method. Also note that from a total of 70 comparisons, TMPA has 65 $wins$ and 5 $ties$ for each error

Datasets	MMRE			MdMRE			Pred(25)		
cocomo81	5	0	0	5	0	0	5	0	0
cocomo81e	5	0	0	5	0	0	5	0	0
cocomo81s	4	1	0	4	1	0	4	1	0
desharnais77	5	0	0	5	0	0	5	0	0
desharnaisL1	5	0	0	5	0	0	5	0	0
desharnaisL2	5	0	0	5	0	0	5	0	0
nasa93	5	0	0	5	0	0	5	0	0
nasa93_center_2	4	1	0	4	1	0	4	1	0
nasa93_center_5	5	0	0	5	0	0	5	0	0
albrecht	3	2	0	3	2	0	3	2	0
finnish	5	0	0	5	0	0	5	0	0
kemerer	4	1	0	4	1	0	4	1	0
maxwell	5	0	0	5	0	0	5	0	0
miyazaki94	5	0	0	5	0	0	5	0	0
TOTAL	65	5	0	65	5	0	65	5	0

Figure 8.2: The **w/t/l** values values of TMPA vs. ABE0 with $k \in \{1,2,3,4,5\}$ on selected datasets. We can see these values of **w/t/l** as the skyline for D-ABE. Note that this figure is Figure 8.1 minus the shallow datasets.

measure. This can be seen as the *skyline* for D-ABE. In other words, assuming that the heuristic approach is as good as the theoretical maximum indicated by TMPA, we would expect a win value of 65.

8.4.2 Static-*k* vs. Dynamic- K (D-ABE)

This section aims to answer the following questions: 1) How better/worse is D-ABE w.r.t. ABE0, i.e. is it worth the effort of designing another heuristic method? 2) After having the *skyline* of TMPA identified in the previous section, how much of this skyline is D-ABE able to cover?

Figure 8.3 shows the actual performance values of D-ABE and ABE0 variants in log-scale. The reason for the use of log-scale is for the purposes of visibility, i.e. to separate points from one another. Note that there are 3 plots (one for each error measure) in Figure 8.3. From these plots, it is easy to observe that TMPA is much better than D-ABE as well as static-ABE0 variants. Furthermore, we can see that D-ABE lies somewhere between TMPA and the best static-ABE0 variant. On the other hand, for some datasets D-ABE and static-ABE0 variants have overlapping performance values.

Although Figure 8.3 is intuitive and helps to observe actual performance values, it is difficult to make a conclusive remark from from that figure. Therefore, we consult to Figure 8.4, where the best performing method according to a particular error measure for a given dataset is marked with the ▲ symbol. In other words, Figure 8.4 acts like a magnifying-glass to Figure 8.3 in the sense that it magnifies the overlapping regions and marks the best performing method.

The *TOTAL* rows (highlighted rows) of Figure 8.4 shows how many times a particular method appeared as the best performer for a each one of the error measures. Since we have 14 datasets, the maximum number of times a method can be the best performer is 14. *For MMRE error measure*, D-ABE outperforms the ABE0 variants in 12 out of 14 cases. The closest followers of D-ABE are ABE0 variants with *1NN* and *4NN*, which both appear only once as the best performer. *For MdMRE*, D-ABE appears 9 times as the best performer, followed by *5NN* which appears 3 times as the best method. That is, D-ABE outperforms ABE0 variants by orders of magnitude. *The scenario for Pred(25)* is similar to that of MMRE and MdMRE: With a sum of 9 out of 14 times, D-ABE outperforms its closest followers (*1NN, 2NN, 4NN* all being the best performer 3 times) by orders of magnitude.

The comparison of performance values helps us reveal the answer to the first question:

> *1) Is it worth the effort of designing another heuristic method?* As we observed from the comparisons, D-ABE attains better performance values than ABE0 variants. In fact, the number of times D-ABE appears to be the best performer is by order of magnitude more when compared to ABE0 methods. So the answer is: Yes, the effort invested in designing the heuristic of D-ABE proves beneficial, hence is worth the effort.

To see how much of the TMPA *skyline* was discovered by D-ABE, we need to consult the **w/t/l** values. Figure 8.5 shows the **w/t/l** values of D-ABE, when compared to ABE0 variants of *1NN, 2NN, 3NN, 4NN* and *5NN*. Note that Figure 8.5 is identical to Figure 8.2, with the difference that Figure 8.2 compares TMPA to static-ABE0 variants, whereas Figure 8.5 compares D-ABE to static-ABE0 variants. The sum of *w/t/l* values for any *dataset&error-measure* intersection is 5. The sum of all **w/t/l** values per error measure over 14 datasets is: *14 datasets* $\times$ *5 comparisons = 70*.

We see from Figure 8.5 that D-ABE has win values of 21, 22, 22 for the error-measures of MMRE, MdMRE and Pred(25) respectively. Out of a total of 70 comparisons, these win values correspond to 30%, 32% and 32% (respectively). Furthermore, note that D-ABE never loses against

ABE0 variants when subject to MdMRE and Pred(25). When subject to MMRE it only loses 1 case out of 70. Figure 8.6 combines the results of Figure 8.5 and Figure 8.2. By using this figure, we can answer the second question: *2) How much of the TMPA-skyline is D-ABE able to cover?* See in Figure 8.6 that the highest percentage of win values that an optimum ABE0 variant can attain against *static-k* variants is 93%. D-ABE is able to discover $30\ to\ 32\%$ of this space. In other words, the proposed heuristic is able to discover around $\frac{1}{3}$ of the space between *static-k* based ABE0 variants and the optimum ABE0 variant (i.e. TMPA).

8.5 Conclusion

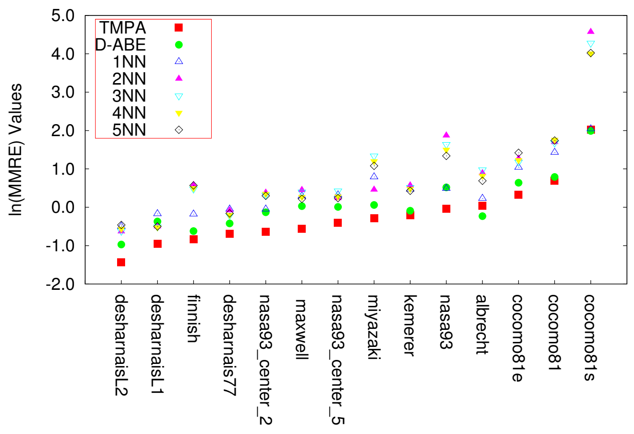
Easy-path supports the following strategy: Identify hard-cases and remove them from the training set. The definition of *hard-case* changes according to context. In the context of this research, hard-cases refer to instances that violate the assumption on which D-ABE was built: *"TMPA indicates the best-set to be retained"*. Since D-ABE is essentially a CBR system, we need a way to identify the training instances that are to be *retained* prior to estimation. We used TMPA for that purpose. After running TMPA on the training instances, we identified the hard cases for which even the TMPA performs badly and *retained* only the *easy-cases*.

We have seen that D-ABE is far better than *static-k* based ABE0 methods, both in mere performance value comparisons and in statistical checks. Also, we saw that D-ABE is able to discover a significant portion ($\frac{1}{3}$) of the performance space that was left to be discovered between static-ABE0 methods and TMPA. Furthermore, easy-path has proven beneficial in another research [16] as well. Therefore, the recommended 8^{th} principle of SEE is:

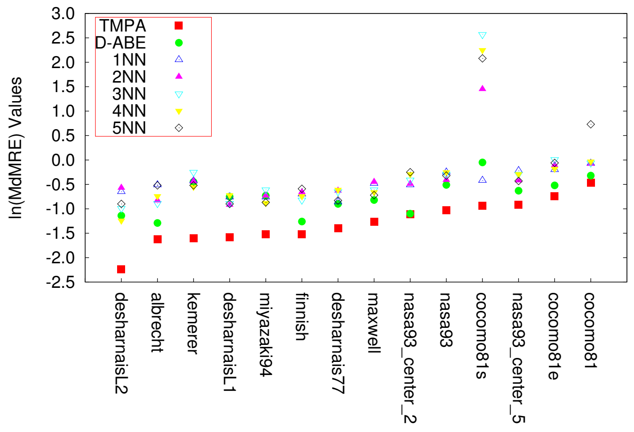
$$\boxed{\text{Principle \#8: Use Easy-path Design}}$$

In summary, the contributions of the research that led to the 8^{th} principle are:

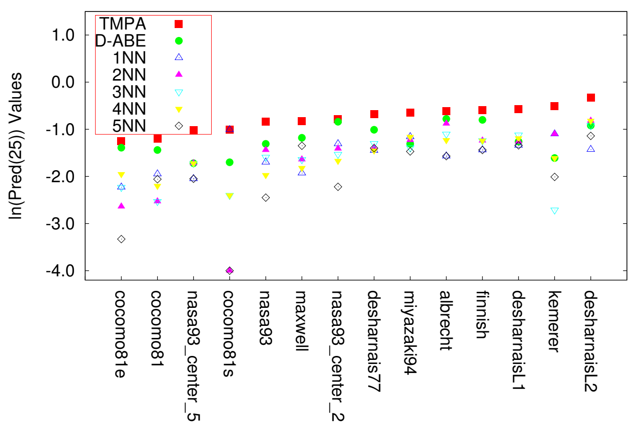
- An ABE design principle that can be applied to different ABE methods.
- Discovering the performance space between *static-k* based ABE methods and TMPA.



(a) Logged and sorted MMRE values



(b) Logged and sorted MdMRE values



(c) Logged and sorted Pred(25) values

Datasets	D-ABE	1NN	2NN	3NN	4NN	5NN
<u>MMRE</u>						
cocomo81	▲					
cocomo81e	▲					
cocomo81s	▲					
desharnais77	▲					
desharnaisL1					▲	
desharnaisL2	▲					
nasa93		▲				
nasa93_center_2	▲					
nasa93_center_5	▲					
albrecht	▲					
finnish	▲					
kemerer	▲					
maxwell	▲					
miyazaki94	▲					
Total	12	1			1	
<u>MdMRE</u>						
cocomo81	▲					
cocomo81e	▲					
cocomo81s		▲				
desharnais77	▲					
desharnaisL1			▲			
desharnaisL2					▲	
nasa93	▲					
nasa93_center_2	▲					
nasa93_center_5	▲					
albrecht	▲					
finnish	▲					
kemerer					▲	
maxwell	▲					
miyazaki94					▲	
Total	9	1	1		3	
<u>Pred(25)</u>						
cocomo81	▲					
cocomo81e	▲					
cocomo81s		▲				
desharnais77	▲					
desharnaisL1				▲		
desharnaisL2			▲		▲	
nasa93	▲					
nasa93_center_2	▲					
nasa93_center_5	▲		▲	▲	▲	
albrecht	▲					
finnish	▲					
kemerer		▲	▲			
maxwell		▲			▲	
miyazaki94	▲					
Total	9	3	3	2	3	

Figure 8.4: Based on the values of Figure 8.3, the method yielding the best performance for each dataset is indicated with a ▲ symbol. At the end of every performance measure a *"Total"* row indicates how many times a method yielded the best performance. Note that the proposed method *D-ABE* attains the best performance many more times than any other static-ABE0 method.

	MMRE			MdMRE			Pred(25)		
Datasets	w	t	l	w	t	l	w	t	l
cocomo81	4	1	0	4	1	0	4	1	0
cocomo81e	1	4	0	1	4	0	1	4	0
cocomo81s	0	5	0	0	5	0	0	5	0
desharnais77	1	4	0	1	4	0	1	4	0
desharnaisL1	0	5	0	0	5	0	0	5	0
desharnaisL2	1	4	0	1	4	0	1	4	0
nasa93	3	1	1	4	1	0	4	1	0
nasa93_center_2	4	1	0	4	1	0	4	1	0
nasa93_center_5	1	4	0	1	4	0	1	4	0
albrecht	1	4	0	1	4	0	1	4	0
finnish	5	0	0	5	0	0	5	0	0
kemerer	0	5	0	0	5	0	0	5	0
maxwell	0	5	0	0	5	0	0	5	0
miyazaki94	0	5	0	0	5	0	0	5	0
TOTAL	21	48	1	22	48	0	22	48	0

Figure 8.5: The **w/t/l** values values of D-ABE vs. ABE0 with $k \in \{1, 2, 3, 4, 5\}$ on targeted datasets. We can see these values of **w/t/l** as the skyline for our hybrid method.

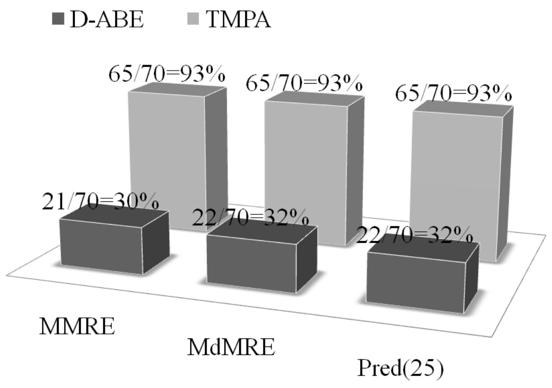


Figure 8.6: The comparison of D-ABE to the skyline proposed by TMPA. The win values of both TMPA and D-ABE, as well as their relative percentages out of 70 comparisons are given in the form of a bar graph. The *skyline* is given by the gray-colored bars and the amount discovered by D-ABE is indicated by the black bars.

Chapter 9

Principle #9, Use Relevancy Filtering

Finding enough training data for local problems is one of the fundamental problems of estimation methods [21–23] and software effort estimation (SEE) is no exception [2, 21]. In this chapter, we argue that finding local training data can be addressed through filtering cross domain data with appropriate methods. We will provide background information regarding earlier work on relevancy-filtering of cross domain data. Then we will introduce our proposed relevancy-filtering method called TEAK. The experimentation reported in this chapter covers a wide range of data sets: Public data sets of Figure 2.1 as well as proprietary data sets of the Tukutuku database (as given in Figure 2.3). Furthermore, we will evaluate the cross domain experiments from the perspective of transfer learning and provide experimentation for the transfer learning experiments across different time-frames.

In all the experiments we have conducted for the research presented in this chapter, we have seen that relevancy filtering works for a big majority of the cross company data experiments. Therefore, the principle we will defend in this chapter will be: "Use Relevancy Filtering" for the local data problems. The rest of this chapter defines the problem, related terms and presents our findings. Finally we conclude this chapter with the contributions of this research.

A company willing to use *"within data"*, a.k.a. local data, will face the problem of considerable time and resource requirement for data collection and maintenance activities. In this research local data refers to a company's own domain data or data from the same time interval. Another problem of local data is aging, i.e. how to choose or disregard projects from older time intervals [127]. We refer to aforementioned problems of within data as "local data issues". The benefit of being able to transfer knowledge of *"cross data"* (the data either coming from another domain or from a different time interval) is twofold: 1) Possibility to cure the local data issues and 2) ability to identify which instances to use from past time intervals. Instance transfer is the process of keeping some of the relevant training instances (while discarding the others).

For the cross domain data transfer research presented in this chapter, we use a novel transfer learning method based on variance-based instance selection, called TEAK [16]. We show that the use of this methods enables successful transfer of data across space (i.e. between organizations) and across time (i.e. between different time intervals). To the best of our knowledge, this is the first report in the SEE literature of a single method that can effectively transfer data across time and space. The proposed method was evaluated on the proprietary data sets of 8 Web companies from the Tukutuku [12] data base (for transfer between domains) as well as publicly available NASA data sets called Cocomo81and Nasa93 (for transfer between time intervals). In the experimentation, each test instance is evaluated in two different scenarios:

- In the first scenario, the test instance is allowed to use only within training data (i.e. restricted to its own domain or time interval).
- In the second scenario, the test instance is allowed to use cross as well as within data (i.e. it is allowed to transfer knowledge from other domains or time intervals).

The performance of the test instances in both scenarios are compared according to 8 different error measures (MAR, MER, MMRE, MdMRE, Pred(25), MBRE, MIBRE and SA) subject to Wilcoxon test (at 95% confidence). In 6 out of 8 companies of the Tukutuku data base, the instance transfer between domains enabled cross data performance to be statistically significantly the same as the performance of within data. In all cases of the transfer learning between time intervals, the within and cross data performance were statistically the same. The second scenario enables us to make a novel contribution to the problem of cross data borders by investigating the *"selection tendency"* of test instances. Selection tendency is defined to be the percentage of instances selected from within and cross data sources. We have found that if a test instance is allowed to use a blend of cross and within data sources (as in the case of the second scenario), test instances select equally likely from within and cross data.

9.1 Transfer Learning

The collection of local data requires allocation of time and resources. To remove the need of local data collection in SEE, various transfer learning studies have been performed [19, 21]. Another reason for the necessity of transfer learning solutions in SEE is the aging of data [128]. In time, data can be outdated and some data instances belonging to an earlier time period may no longer be representative of the current trends. Our study proposes a transfer learning solution that can work both for domain as well as time period changes.

A learning problem can be defined by:

- a specific domain D, which consists of a feature space and a marginal distribution defining this space;
- and a task T, which is the combination of a label space and an objective estimation function.

Transfer learning allows for the training and test data to have different domains and tasks [129]. According to Jialin et al. transfer learning can be formally defined as follows [128]: Assuming we have a source domain D_S, a source task T_S, a target domain D_T and a target task T_T; transfer learning tries to improve an estimation method in D_T using the knowledge of D_S and T_S. Note that the assumption in the above definition is that $D_S \neq D_T$ and $T_S \neq T_T$. There are various subgroups of transfer learning, which define the relationship between traditional machine learning methods and various transfer learning settings, e.g. see Table 1 of [128]. SEE cross data experiments have the same task but different domains, which places them under the category of transductive transfer learning [130]. Note that in this research a domain refers to particular companies within Tukutuku data base, however that does not mean that domain always refers to organizations. The definition of a domain is context dependent, i.e. whereas in this research it refers to particular companies, in another study it may refer to certain software projects, certain business domains etc.

There are 4 different approaches to transfer learning [128]: instance-transfer (or instance-based transfer) [131], feature representation transfer [132], parameter-transfer [133] and relational-knowledge transfer [134]. The transfer learning approach of the estimation method used in this research corresponds to instance-transfer. The benefits of instance-transfer learning are used in various research areas, e.g. Ma et al. use transfer learning for cross-company defect prediction, where they use a weighted Naive Bayes classifier [129]. Other research areas that benefit from instance-transfer are text classification [135], e-mail filtering [136] and image classification [137].

9.2 Instability of Transfer Learning Studies in SEE

A standard assumption in many model-based estimation methods is that it is necessary and useful to process *all* the available training data points. The empirical lessons learned from *instance selection* and *irrelevancy removal* say that it is not the case. The results of the experiments reported in this chapter, as well as prior work [2, 19, 22, 23], are also along the same lines: Use of all the training data from a cross data source is not useful; on the contrary, using relevancy filtering to filter out irrelevant cross data training instances increases the performance of cross data.

Previous research focuses on 2 fundamental problems of cross data usage [2,21]: 1) Cross data performance, when compared to within data performance and 2) defining cross data borders. Much of the research focuses on the first problem [21, 22]. The second problem emerges upon recent evidence that disagrees with the notion that features like domain or time interval define strict cross and within data borders, hence disrupts knowledge transfer [2, 19]. As part of the research that led to this chapter, we authored a manuscript, which showed that transfer learning (through instance transfer) helps cross data performance on public SEE data sets [19]. On a follow up study of this manuscript [19] we aimed to tackle the second problem of cross data (i.e. cross data borders), we found that using single features like domain (e.g. geographical location, company name) to define strict cross data borders is misleading [2] and knowledge can be transferred through these borders via instance transfer methods.

Transferring knowledge between cross borders of different time intervals has been paid very little attention. To the best of our knowledge, the only work that has previously questioned transfer learning between different time frames is that of Lokan et al. [138, 139]. In [138] Lokan and Mendes found out by using chronological sets of instance in a cross-company learning setting that time frame divisions of instances did not affect prediction accuracy. In [139], they found out that it is possible to suggest a window size of a time frame of past instances, which can yield performance increase in estimation. They also note that the size of the window frame is data set dependent. Our research builds on the prior findings to provide evidence of knowledge transfer through both domain and time.

The prior results on the performance of cross data in transfer learning are unstable. In their review, Kitchenham et al. [21] found equal evidence for and against the value of transfer learning in SEE. Among 7 studies reviewed by Kitchenham et al., 4 studies favored within data, whereas 3 found that cross data is not statistically significantly worse than within data. In the field of defect prediction, Zimmermann et al. studied the use of cross data [23]. Zimmermann et al. found that predictors performed worse when trained on cross-application data than from within-application data. From a total of 622 cross and within data comparisons, they report that within outperformed cross in 618 cases. Recently Ma et al. defined the cross data learning problem in the research field of defect prediction as a transfer learning problem [129]. Ma et al. propose a Naive Bayes variant, so called Transfer Naive Bayes (TNB), so as to use all the appropriate features from the training data. TNB is proposed as an alternative transfer learning method for defect prediction when there are too few training data. According to Ma et al. cross data learning problem of defect prediction corresponds to an inductive transfer learning setting; where source and target tasks are the same, yet source and target domains are different. The inductive transfer learning methods are summarized as either instance transfer or feature transfer [140]. The current literature of cross data learning in defect prediction as well as SEE focuses on instance transfer.

Turhan et al. compared defect predictors learned from cross or within data. Like Zimmermann et al., they found that using *all* cross resource data leads to poor estimation method performance (very large false alarm rates). However, after *instance selection* pruned away irrelevant cross resource data, they found that the cross resource estimators were equivalent to the estimators learned from within resource data [22]. Motivated by Turhan et al. [22], Kocaguneli et al. [19] used *instance selection* as a pre-processor for a study of transfer learning in SEE, where test instances are allowed to use only cross or only within data. In a limited study with three data sets, they found that through instance selection, the performance differences in the predictors learned from cross or within data were not statistically significant. This limited study was challenged by Kocaguneli et al. in another study that uses 8 different data sets [2]. The results were identical: performance differences of within and cross data are not significant.

9.3 Resolving Instability

Conflicting results are nothing new for SEE. As we noted in the previous section of this chapter, there are different studies reporting conflicting results on the use of cross company data. The problem of conflicting results is attracting attention from the SEE community and a promising progress can be seen towards that issue. For example, in Chapter 5 we were able to see that we can elicit superior solo-methods, provided that we investigate a large corpus of SEE methods, datasets and error measures. Hence, this chapter challenges prior results in transfer learning studies in SEE, on a large number of data sets, which include public as well as proprietary data sets. Note that one of the major results of Chapter 5 is that all the superior 13 solo-methods use CART or $k = 1$ nearest -neighbor. This is significant since both these estimators use multiple features to sub-divide the training data:

- k-th nearest neighbor algorithms use all project features (perhaps, weighted by some feature) to determine related projects [38];
- Tree-based algorithms, like CART [48], divide the data into multiple branches, where each branch tests and divides that data on multiple features.

9.3.1 TEAK

TEAK is a *variance-based* instance selector that discards training data associated with regions of high dependent variable (effort) variance [16]. TEAK is based on the locality principle, which states that instances that are close to one another in space according to a distance measure (e.g. Euclidean distance measure) are similar instances and should have similar dependent variable values. A high variance region, where similar instances have very different effort values (hence the high variance) violates the locality assumption and is pruned away by TEAK [16]. TEAK augments ABE0 with instance selection and an indexing scheme for filtering relevant training examples. In summary, TEAK is a two-pass system:

- Pass 1 prunes training instances implicated in poor decisions (instance selection);
- Pass 2 retrieves closest instances to the test instance (instance retrieval).

In the first pass, training instances are combined using greedy-agglomerative clustering (GAC), to form an initial cluster tree that we call GAC1; e.g. see Figure 9.1. Level zero of GAC1 is formed by leaves, which are the individual project instances. These instances are greedily combined (combine the two closest instances) into tuples to form the nodes of upper levels. The variance of the effort values associated with each sub-tree (the performance variance) is then recorded and normalized: $min..max$ to $0..1$. The high variance sub-trees are then pruned, as these are the sub-trees that would cause an ABE method to make an estimate from a highly variable instance space. Hence, pass one prunes sub-trees with a variance in the vicinity of $rand() * \alpha\%$ of the maximum vari-

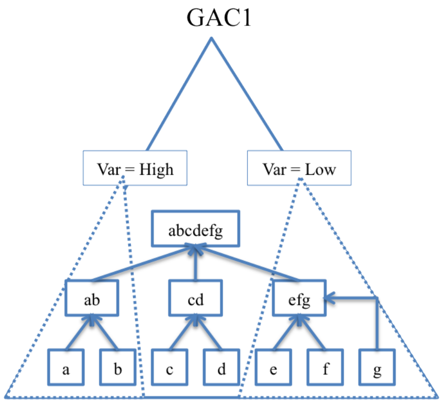


Figure 9.1: A sample GAC tree with regions of high and low variance (dashed triangles). GAC trees may not always be binary. For example here, leaves are odd numbered, hence node "g" is left *behind*. Such instances are pushed *forward* into the closest node in the higher level. For example, "g" is pushed forward into the "e+f" node to make "e+f+g" node.

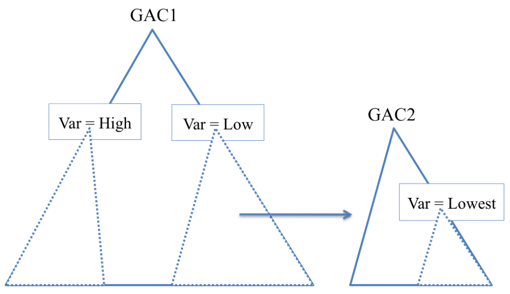


Figure 9.2: Execution of TEAK on 2 GAC trees, where tree on the left is GAC1 of Figure 9.1 and the one on the right is GAC2. The instances in the low variance region of GAC1 are selected to form GAC2. Then test instance traverses GAC2 until no decrease in effort variance is possible. Wherever the test instance stops is retrieved as the subtree to be used for adaptation (var=lowest labeled, dashed triangle of GAC2).

ance seen in any tree, where $rand()$ gives a normal random value from 0-1 interval. After some experimentation, we found that $\alpha = 10$ leads to estimates with lowest errors.

The leaves of the remaining sub-trees are the *survivors* of pass one. They are filtered to pass 2 where they are used to build a second GAC tree (GAC2). GAC2 is generated in a similar fashion to GAC1, then it is traversed by test instances that are moved from root to leaves. Unlike GAC1, this time variance is a decision criterion for the movement of test instances: If the variance of the current tree is larger than its sub-trees, then continue to move down; otherwise, stop and retrieve the instances in the current tree as the analogies. TEAK is a form of ABE0, so its adaptation method is the same, i.e. take the median of the analogy effort values. A simple visualization of this approach is given in Figure 9.2.

We use TEAK in this study since, as shown by the leave-one-out experiments of Kocaguneli et al. [16], its performance is comparable to other commonly-used effort estimators including neural networks (NNet) and linear regression (LR). For a complete analysis of TEAK compared to NNet,

					Simple ABE0				
	TEAK	LR	NNet	*k=best*	k=1	k=16	k=2	k=4	k=8
MdMRE									
Cocomo81	▲								
Cocomo81e	▲								
Cocomo81o	▲								
Nasa93		▲							
Nasa93c2		▲							
Nasa93c5	▲								
Desharnais		▲							
Sdr	▲								
ISBSG-Banking	▲								
Count	6	3	0	0	0	0	0	0	0
Pred(25)									
Cocomo81	▲								
Cocomo81e			▲						
Cocomo81o	▲								
Nasa93		▲							
Nasa93c2		▲							
Nasa93c5	▲								
Desharnais		▲							
Sdr	▲								
ISBSG-Banking	▲								
Count	5	3	1	0	0	0	0	0	0
MAR									
Cocomo81	▲								
Cocomo81e	▲								
Cocomo81o	▲								
Nasa93		▲							
Nasa93c2		▲							
Nasa93c5	▲								
Desharnais		▲							
Sdr	▲								
ISBSG-Banking	▲								
Count	6	3	0	0	0	0	0	0	0

Figure 9.3: Results from 20 repeats of a leave-one-out experiment, repeated for the performance measures of MdMRE, Pred(25) and MAR. Black triangles mark when an estimator was one of the top-ranked methods for a particular data set (where ranking was computed via $win - loss$ from a Wilcoxon test, 95% confidence). The $Count$ rows show the number of times a method appeared as the top performing variant. The results of this figure come from [16].

LR as well as various ABE0 methods please refer to Figure 7 of [16]. Figure 9.3 can be interpreted as follows:

- The columns $k = 1, 2, 4, 8, 16$ denote variants of standard ABE0 where estimates are generated from the k-th nearest neighbors.
- The column $k = best$ denote a variant of ABE0 where k was chosen by an initial preprocessor that chose a best k value after exploring the training data.
- The columns LR and $NNet$ refer to linear regression and neural nets.

The black triangles in Figure 9.3 mark when an estimator was one of the top-ranked methods for a particular data set. Ranking was accomplished via the $win - loss$ calculation. The key feature of Figure 9.3 is that TEAK always performed better than the other ABE0 methods, and usually performed better than neural nets. TEAK's only near-rival was linear regression but, as shown in the LR columns, TEAK was ranked top nearly twice as much as linear regression.

9.4 Experimentation

The goals of the experiments carried out in this chapter can be summarized as follows:

1. Compare the performance of TEAK when trained from just within data versus when trained from a combination of cross and within data.
2. The *retrieval tendency* goals question the tendency of a *within* test instance to retrieve *within* or *cross* data. In other words, given the chance that a test instance had access to *within* and *cross* data at the same time, what percentage of every subset would be retrieved into *k* analogies used for estimation?

The first goal challenges the findings from older publicly available SEE data sets on the proprietary datasets of 8 different contemporary Web development companies. This is quite important, since the results pertaining to our first goal can give the practitioners an idea about how well cross data studies may perform in their actual settings. The second goal challenges our assumptions regarding the definition of cross data, i.e. is it right to draw a border and say some data is cross or within based on single features like the company name? The results regarding the second goal are indicators for practitioners regarding which factors to consider and which factors to ignore when using cross data.

9.4.1 Performance Comparison

With regard to performance comparison we have two settings: *Within* and *cross*. In the *within* data setting, only the *within* source is used as the dataset, and a testing strategy of leave-one-out cross-validation (LOOCV) is employed.

Cross data setting uses one instance at a time (since we use LOOCV) from the *within* data as the test set and the *combination of remaining within instances and all the cross* data as the training set. In this setting TEAK derives an estimate for each test instance by adapting the analogies of the training set. Ultimately we end up with T predictions adapted from the training set. Finally, the performances under *within* and *cross* data settings are compared. For that purpose, we use both mere performance values as well as win-tie-loss statistics.

9.4.2 Retrieval Tendency

For retrieval tendency experiments we mark every *within* and *cross* instance in the training set and let the test instance choose analogies from the training data set of within and cross data instances. Note that retrieved analogies are the unique training instances in the lowest-variance region of GAC2 (see Figure 9.2). In this setting our aim is to see what percentage of *within* and *cross* subsets would appear among retrieved *k* analogies. The retrieval percentage is the average (over all test instances) ratio of instances retrieved in analogies to the total size of the training set:

$$Percentage = \frac{NumberOfRetrievedAnalogies}{TrainingSetSize} * 100 \tag{9.1}$$

Dataset	W			MAR	
	Win	Tie	Loss	W	C
tuku1	8	12	0	27.8	55.9
tuku2	20	0	0	6.1	60.6
tuku3	20	0	0	661.7	2577.6
tuku4	2	18	0	104.5	177.4
tuku5	0	20	0	305.8	315.4
tuku6	8	12	0	25.8	42.7
tuku7	2	18	0	542.7	551.6
tuku8	1	19	0	87.6	103.0

Dataset	W			MMRE	
	Win	Tie	Loss	W	C
tuku1	9	11	0	0.9	3.1
tuku2	20	0	0	1.1	20.6
tuku3	20	0	0	0.3	0.9
tuku4	2	18	0	0.4	5.5
tuku5	0	19	1	2.9	1.0
tuku6	5	15	0	0.3	0.7
tuku7	3	17	0	1.2	1.0
tuku8	0	20	0	0.9	0.8

Dataset	W			MdMRE	
	Win	Tie	Loss	W	C
tuku1	8	12	0	0.6	1.0
tuku2	20	0	0	0.7	10.2
tuku3	20	0	0	0.2	1.0
tuku4	2	18	0	0.3	0.9
tuku5	0	19	1	0.9	0.8
tuku6	5	15	0	0.3	0.4
tuku7	3	17	0	0.6	0.7
tuku8	0	20	0	0.4	0.6

Dataset	W			Pred(25)	
	Win	Tie	Loss	W	C
tuku1	6	10	4	0.4	0.1
tuku2	18	0	2	0.1	0.2
tuku3	20	0	0	0.7	0.4
tuku4	2	18	0	0.5	0.5
tuku5	1	19	0	0.2	0.2
tuku6	5	15	0	0.4	0.4
tuku7	3	17	0	0.2	0.3
tuku8	0	20	0	0.4	0.4

Figure 9.4: Performance comparison of within (**W**) and cross (**C**) data w.r.t. 4 of 8 different performance measures: MAR, MMRE, MdMRE, Pred(25). Win, tie, loss values are w.r.t. **W**. Last two columns are the actual performance measure values of **W** and **C**.

9.5 Results

9.5.1 Performance Comparison

Comparing the performance of within and cross data is the first goal of this experimentation. Due to space constraints and also in order to make the results more readable we equally divided the domain transfer results of the Tukutuku subsets into two figures: Figure 9.4 and Figure 9.5. The time interval transfer results of Cocomo81 and Nasa93 are provided in Figure 9.7 and Figure 9.8, respectively.

Figure 9.4 shows a uniformity of results. The tie values are very high for 5 out of 8 companies (tuku1, tuku4-to-7), which means that cross data performance of is as good as within data performance. The high tie values are also reflected in the actual error measures. See that the values of the error measures (last two columns of Figure 9.4) for within (**W**) and cross (**C**) data sources are very close to one another for tuku1 and tuku-4-to-7. For 1 company, tuku8, within and cross data performance depends on the error measure: According to all error measures except MMER the within and cross performances are very close, whereas for MMRE the within data performance appears to be better. For 2 companies out of 8 (tuku2 and tuku3), the within data performance is dominantly better than cross data performance with a win value of 20. Remember from §9.3.1 that TEAK performs 20 times LOOCV, hence the total of win, tie and loss values for each data set subject to each error measure amounts to 20. The reason of 20 LOOCV repeats is to remove the bias due to the random pruning step of TEAK.

Figure 9.5 shows the within and cross data performance of TEAK for the error measures of MMER, MBRE, MIBRE and SA. The reading of Figure 9.5 is exactly the same as of Figure 9.4, i.e. it shows the win, tie, and loss values according to 4 error measures as well as the actual error measure values. The general pattern we have observed from 4 error measures in Figure 9.4 are also visible in Figure 9.5. In relation to the error measures MBRE, MIBRE and SA, in 6 data sets (tuku1, tuku2, tuku4-to-7) cross and within performances are the same. According to the MMER, within performance is better than the cross performance for 2 data sets: tuku3 and tuku8.

The aforementioned results support the prior results reported by Kocaguneli et al. [2], where they have used 21 public data sets from PROMISE data repository. A summary of their results

Dataset	**W**			**MMER**	
	Win	**Tie**	**Loss**	**W**	**C**
tuku1	0	20	0	1.0	1.2
tuku2	0	20	0	1.6	0.8
tuku3	20	0	0	0.3	58.2
tuku4	0	20	0	4.9	4.9
tuku5	6	14	0	2.3	8.2
tuku6	8	12	0	0.4	2.5
tuku7	6	14	0	5.1	9.2
tuku8	14	6	0	0.7	2.5

Dataset	**W**			**MBRE**	
	Win	**Tie**	**Loss**	**W**	**C**
tuku1	6	14	0	1.4	3.8
tuku2	20	0	0	2.2	20.6
tuku3	20	0	0	0.3	58.2
tuku4	1	19	0	4.9	9.7
tuku5	0	20	0	4.6	8.5
tuku6	6	14	0	0.4	2.8
tuku7	5	15	0	5.7	9.6
tuku8	2	18	0	1.2	2.7

Dataset	**W**			**MIBRE**	
	Win	**Tie**	**Loss**	**W**	**C**
tuku1	2	18	0	0.5	0.6
tuku2	17	3	0	0.5	0.8
tuku3	20	0	0	0.2	0.9
tuku4	1	19	0	0.4	0.8
tuku5	0	20	0	0.6	0.6
tuku6	6	14	0	0.3	0.4
tuku7	4	16	0	0.5	0.6
tuku8	3	17	0	0.4	0.5

Dataset	**W**			**SA**	
	Win	**Tie**	**Loss**	**W**	**C**
tuku1	8	12	0	0.2	-0.7
tuku2	20	0	0	-0.0	-10.3
tuku3	20	0	0	0.0	-2.7
tuku4	2	18	0	0.4	0.1
tuku5	0	20	0	0.1	0.0
tuku6	8	12	0	0.2	-0.5
tuku7	2	18	0	0.3	0.3
tuku8	1	19	0	-0.1	-0.4

Figure 9.5: Performance comparison of within (represented with **W**) and cross (represented with **C**) data w.r.t. 4 of 8 different performance measures: MMER, MBRE, MIBRE, SA.

on 21 public data sets are provided in Figure 9.6. As can be seen in Figure 9.6, Kocaguneli et al. uses 4 error measures and identifies only 2 data sets (gray highlighted rows) for which within data

Dataset	MAR			MMRE			MdMRE			Pred(30)		
	Win	Tie	Loss	Win	Tie	Loss	Win	Tie	Loss	Win	Tie	Loss
cocomo81e	0	20	0	0	16	4	4	16	0	4	16	0
cocomo81o	0	20	0	2	18	0	2	18	0	2	18	0
cocomo81s	18	2	0	15	5	0	15	5	0	13	5	2
nasa93_center_1	0	20	0	0	20	0	0	20	0	0	20	0
nasa93_center_2	4	16	0	2	18	0	2	18	0	2	18	0
nasa93_center_5	0	20	0	0	12	8	8	12	0	8	11	1
desharnaisL1	11	9	0	9	11	0	9	11	0	9	11	0
desharnaisL2	0	20	0	0	20	0	0	20	0	0	20	0
desharnaisL3	0	20	0	2	18	0	2	18	0	2	18	0
finnishAppType1	0	20	0	0	20	0	0	20	0	0	20	0
finnishAppType2345	0	20	0	0	17	3	0	17	3	0	17	3
kemererHardware1	0	0	20	0	0	20	0	0	20	0	0	20
kemererHardware23456	0	20	0	0	20	0	0	20	0	0	20	0
maxwellAppType1	6	14	0	1	19	0	1	19	0	0	19	1
maxwellAppType2	0	18	2	0	19	1	0	19	1	0	19	1
maxwellAppType3	0	20	0	1	19	0	1	19	0	1	19	0
maxwellHardware2	0	20	0	0	20	0	0	20	0	0	20	0
maxwellHardware3	0	20	0	0	20	0	0	20	0	0	20	0
maxwellHardware5	0	20	0	0	20	0	0	20	0	0	20	0
maxwellSource1	6	14	0	1	19	0	1	19	0	1	19	0
maxwellSource2	0	20	0	0	20	0	0	20	0	0	20	0

Figure 9.6: Summary of prior TEAK results [2] on 21 public data sets. For 19 data sets, cross and within data performances are the same (note high tie values). For only 2 data sets (highlighted) within data performance is better than cross.

performance is worse than cross company. For $21 - 2 = 19$ data sets, cross data performance is statistically significantly the same as that of within company.

Figure 9.7 and Figure 9.8 show the performance results of transfer learning in time intervals for Cocomo81 and Nasa93. For Cocomo81, two within sources are defined: 1) projects developed from 1960 to 1975 (called as coc-60-75) and 2) projects developed from 1976 onwards (called as coc-76-rest). Similarly, the subsets of Nasa93 are: 1) projects from 1970 to 1979 (called as nasa-70-79) and 2) projects from 1980 onwards (called as nasa-80-rest). In both Figure 9.7 and Figure 9.8, the tie values are quite high with the smallest tie value of 16. Note that in none of the two figures there is a highlighted row, which means that in none of the time interval instance transfer experiments was there a case where TEAK failed to transfer instances between different time intervals. The implications of within and cross data experiments through instance transfer in time intervals are important for practitioners. Transfer learning results on Cocomo81 and Nasa93 subsets show that instance transfer methods, such as TEAK, may help companies use aged data sets. Figure 9.7 and Figure 9.8 fundamentally show that decades of time difference can be crossed

Dataset	**W**			**MAR**	
	Win	**Tie**	**Loss**	**W**	**C**
coc-60-75	0	20	0	1181.0	1194.1
coc-76-rest	0	20	0	383.6	385.5

Dataset	**W**			**MMRE**	
	Win	**Tie**	**Loss**	**W**	**C**
coc-60-75	0	19	1	2.3	1.2
coc-76-rest	0	20	0	1.9	1.6

Dataset	**W**			**MdMRE**	
	Win	**Tie**	**Loss**	**W**	**C**
coc-60-75	0	19	1	0.9	0.9
coc-76-rest	0	20	0	0.8	0.8

Dataset	**W**			**Pred(25)**	
	Win	**Tie**	**Loss**	**W**	**C**
coc-60-75	0	19	1	0.1	0.0
coc-76-rest	0	20	0	0.1	0.1

Dataset	**W**			**MMER**	
	Win	**Tie**	**Loss**	**W**	**C**
coc-60-75	0	20	0	34.4	47.6
coc-76-rest	0	20	0	8.9	8.8

Dataset	**W**			**MBRE**	
	Win	**Tie**	**Loss**	**W**	**C**
coc-60-75	1	19	0	35.9	48.1
coc-76-rest	0	20	0	10.2	9.7

Dataset	**W**			**MIBRE**	
	Win	**Tie**	**Loss**	**W**	**C**
coc-60-75	0	19	1	0.8	0.7
coc-76-rest	0	20	0	0.6	0.6

Dataset	**W**			**SA**	
	Win	**Tie**	**Loss**	**W**	**C**
coc-60-75	0	20	0	0.3	0.2
coc-76-rest	0	20	0	0.2	0.2

Figure 9.7: Performance comparison of Cocomo81 subsets for transfer learning in time.

with the help of instance transfer.

The results of this research on proprietary data sets combined with the prior results of public data sets [2] give us a broader picture of the cross data performance. By using instance transfer methods, such as TEAK between domains, we have:

- 5 out of 8 proprietary data sets;

Dataset	**W**			**MAR**	
	Win	**Tie**	**Loss**	**W**	**C**
nasa-70-79	0	20	0	640.0	744.5
nasa-80-rest	4	16	0	339.6	377.6

Dataset	**W**			**MMRE**	
	Win	**Tie**	**Loss**	**W**	**C**
nasa-70-79	0	20	0	3.8	2.2
nasa-80-rest	4	16	0	1.1	2.0

Dataset	**W**			**MdMRE**	
	Win	**Tie**	**Loss**	**W**	**C**
nasa-70-79	0	20	0	0.7	0.8
nasa-80-rest	4	16	0	0.6	0.7

Dataset	**W**			**Pred(25)**	
	Win	**Tie**	**Loss**	**W**	**C**
nasa-70-79	0	20	0	0.2	0.2
nasa-80-rest	4	16	0	0.3	0.2

Dataset	**W**			**MMER**	
	Win	**Tie**	**Loss**	**W**	**C**
nasa-70-79	4	16	0	2.3	4.4
nasa-80-rest	0	20	0	2.7	2.5

Dataset	**W**			**MBRE**	
	Win	**Tie**	**Loss**	**W**	**C**
nasa-70-79	2	18	0	5.6	5.9
nasa-80-rest	2	18	0	3.2	4.0

Dataset	**W**			**MIBRE**	
	Win	**Tie**	**Loss**	**W**	**C**
nasa-70-79	2	18	0	0.6	0.6
nasa-80-rest	2	18	0	0.5	0.6

Dataset	**W**			**SA**	
	Win	**Tie**	**Loss**	**W**	**C**
nasa-70-79	0	20	0	0.2	0.1
nasa-80-rest	4	16	0	0.2	0.1

Figure 9.8: Performance comparison of Nasa93 subsets for transfer learning in time.

- 19 out of 21 public data sets;

where the performance difference between within and cross data is not statistically significant. This shows us that from a total of $21+8=29$ public and proprietary data sets, cross data performs as well as within data for $5+19=24$ cases. Also, by using TEAK for transfer learning between time intervals, we have 2 data sets subject to 8 error measures ($2\times 8=16$ cases), where cross data performance is always the same as within data performance.

9.5.2 Inspecting Selection Tendencies

The second goal of our experimentation is to observe the selection tendencies of the test instances. Figure 9.9 shows what percentage of instances are selected from within data sources (diagonal cells) as well as cross data sources (off diagonal cells) in transfer learning experiments between domains. The first column of Figure 9.9 shows the within data sources (8 different companies) as well as their sizes in parenthesis. The second column shows the number of analogies retrieved from GAC2 on average over 20 runs. For each row, the columns *tuku1-to-8* show how the number of analogies in the second column is distributed to each data source. The values outside the parenthesis in each cell of columns *tuku1-to-8* are the number of analogies selected from that data source; the percentage value of that number w.r.t. the second column is given inside the parenthesis. Figure 9.10 and Figure 9.11 show the selection tendency of test instances for Cocomo81 and Nasa93 time interval transfer learning experiments. These figures are structured in the same manner as Figure 9.9.

Dataset	**# of Analogies**	**tuku1**	**tuku2**	**tuku3**	**tuku4**	**tuku5**	**tuku6**	**tuku7**	**tuku8**
tuku2 (20)	11.0	1.1 (8.0)	1.6 (8.2)	0.3 (2.0)	0.8 (12.9)	0.7 (5.0)	0.7 (8.3)	2.7 (8.7)	3.1 (17.2)
tuku3 (15)	7.3	0.9 (6.2)	1.3 (6.3)	0.4 (2.9)	0.2 (4.1)	0.8 (6.4)	0.2 (2.9)	1.5 (4.9)	1.9 (10.5)
tuku4 (6)	6.7	0.6 (4.4)	1.5 (7.6)	0.2 (1.5)	0.3 (5.0)	0.2 (1.7)	0.7 (8.9)	1.2 (3.8)	2.0 (10.9)
tuku5 (13)	9.3	2.4 (17.2)	1.4 (7.2)	0.3 (2.1)	0.5 (8.5)	0.5 (4.2)	0.4 (5.4)	1.3 (4.0)	2.4 (13.1)
tuku6 (8)	8.9	0.7 (4.7)	1.6 (7.8)	0.2 (1.5)	0.7 (12.4)	0.7 (5.5)	0.7 (8.4)	1.8 (5.7)	2.6 (14.3)
tuku7 (31)	7.8	1.2 (8.3)	1.3 (6.4)	0.3 (2.3)	0.5 (8.4)	0.6 (4.5)	0.1 (1.5)	1.7 (5.5)	2.1 (11.6)
tuku8 (18)	6.9	1.1 (7.9)	1.1 (5.7)	0.3 (2.3)	0.3 (4.5)	0.7 (5.5)	0.3 (3.3)	1.3 (4.3)	1.8 (9.7)

Figure 9.9: The amount of instances selected from within and cross company datasets. The first column is the subset names and their sizes in parenthesis. The second column is the average number of retrieved instances in GAC2. The following columns show the number of analogies from each particular subset, in parenthesis the percentage values of these numbers w.r.t. the second column are given.

The selection tendency values of Tukutuku, Cocomo81 and Nasa93 subsets provide us with suggestions regarding how much data a test instance uses from within and cross data sources during the domain and time interval transfer of instances. From the selection tendency figures, we can

Dataset	# of Analogies	coc-60-75	coc-76-rest
coc-60-75 (20)	3.6	**0.8 (4.1)**	2.8 (6.4)
coc-76-rest (43)	4.5	1.3 (6.6)	**3.2 (7.4)**

Figure 9.10: The amount of instances selected from within and cross company datasets of Cocomo81.

Dataset	# of Analogies	nasa-70-79	nasa-80-rest
nasa-70-79 (39)	7.3	**2.6 (6.6)**	4.7 (8.7)
nasa938089 (54)	9.5	3.2 (8.2)	**6.3 (11.7)**

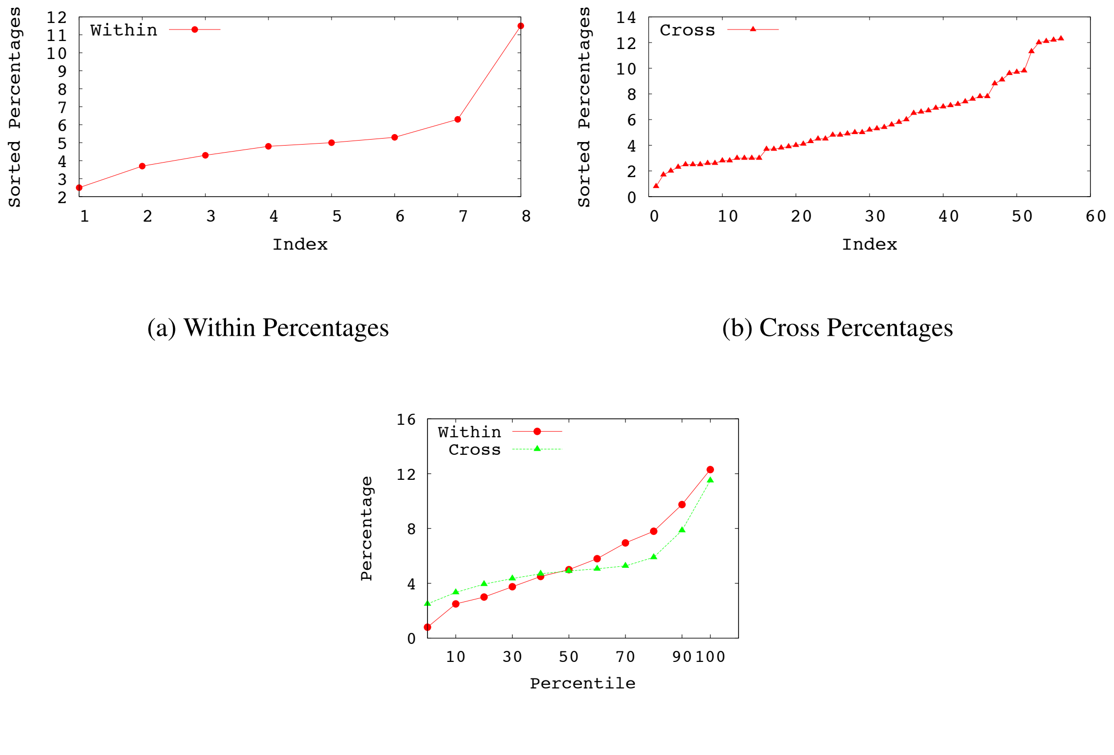
Figure 9.11: The amount of instances selected from within and cross company datasets of Nasa93.

identify two findings:

Finding #1: See the second columns of Figure 9.9, Figure 9.10 and Figure 9.11 that only a very small portion of all the available data (cross and within) is transferred as useful analogies. This finding points to the importance of: a) instance transfer, a.k.a filtering; b) estimation methods like TEAK that are capable of transferring relevant analogies between domains and time intervals. The low number of instances selected are also supported by relevant literature: Chang's prototype generators [141] replaced training sets of size $T = (514, 150, 66)$ with prototypes of size $N = (34, 14, 6)$ (respectively). That is, prototypes may be as few as $\frac{N}{T} = (7, 9, 9)\%$ of the original data. Note that these values are close to how many instances were retrieved in the above results.

Finding #2: When we compare the diagonal and off-diagonal percentages, we see that the values are very close. This is consistent with what Kocaguneli et al. had reported from public data sets [2]. The second finding bears importance regarding the definition of cross data. It shows that defining cross data on the basis of a single feature -e.g. w.r.t. a particular company- is misguided. It also shows that prior findings of domain transfer experiments on the public data sets [2] are stable for proprietary data sets as well as for time interval transfer. TEAK, being an instance transfer method using all features to transfer analogies, selects close amounts of instances from each data source.

To better observe how close within and cross data percentages are, we plot the percentage values of within and cross data sources of Tukutuku subsets in Figure 9.12(a) and Figure 9.12(b), respectively. Figure 9.12(a) and Figure 9.12(b) are basically the plots of diagonal and off-diagonal percentage values of Figure 9.9. We see from Figure 9.12(a) and Figure 9.12(b) that the maximum and minimum percentages of within and cross data selection are very close. To align these percentages, we took the percentile values from 0^{th} percentile to 100^{th} percentile with increments of 10. The resulting percentiles are shown in Figure 9.12(c). See in Figure 9.12(c) that within and cross data have similar percentile values, i.e. the selection tendencies from within and cross sources are very close to one another. Note that percentage and percentile plots are unnecessary for Figure 9.10 and Figure 9.11, since the closeness of within and cross data selection percentages of Cocomo81 and Nasa93 subsets can easily be verified with manual inspection: For Cocomo81 subsets the biggest percentage difference is $6.6 - 4.1 = 2.5\%$; for Nasa93 subsets it is $11.7 - 8.2 = 3.5\%$.

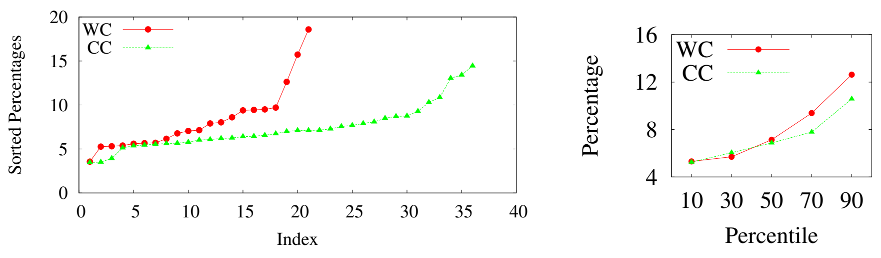


(a) Within Percentages

(b) Cross Percentages

(c) Within and Cross Percentiles

Figure 9.12: Percentages of retrieved instances (a.k.a. analogies) from within (a) and cross (b) data sets. The percentage values come from Figure 9.9. Within company percentages are the gray-highlighted diagonal cells, whereas cross company percentages are the remaining off-diagonal cells. The percentile graph (c) shows the percentiles of (a) and (b).



(a) Percentages (b) Percentiles

Figure 9.13: Percentages and percentiles of instances retrieved by TEAK from within and cross datasets as given by Kocaguneli et al. [2]. Note the similarity of the percentile plot of this figure to that of Figure 9.12.

Figure 9.13 is taken from our manuscript published as part of the research presented in this chapter, regarding the selection tendency experiments on the public data sets [2]. In the performance experiments, we have seen the similarity between the results of this research and that of Kocaguneli et al. in terms of performance. Comparison of Figure 9.13 to Figure 9.12 shows that the similarity of results are also valid in terms of the selection tendencies. See in particular the percentile values of Figure 9.12 and Figure 9.13 that transfer learning between domains of proprietary data sets and public data sets have similar selection tendencies.

9.6 Discussion

Based on the majority of the companies in the domain transfer learning experiments (depending on the error measure, 5 or 6 companies out of 8) the cross data performance is the same as the within data performance. In terms of time interval transfer learning, in *all* of the cases, within and cross data performances were statistically the same. This shows us that instance transfer methods like TEAK that filter cross data can help transfer learning between domains and time intervals so that cross data can perform as well as within data. However, for a minority of the companies in the domain transfer learning experiments, the cross data performance may be far from satisfactory. The reason for the difference of these companies may be hidden in their within data quality, but this statement is just our speculation. For concrete reasons leading to failure of cross data in certain companies, further research is needed on the features defining within and cross data borders.

Another interesting fact is that error measures can result in different conclusions. For example, see in Figure 9.4 and Figure 9.5 that cross data performance for the company tuku8 depends on error measures. This disagreement may cause different companies to make different conclusions depending on the particular error measures they are using.

Note that in the experiments of this chapter, the cross borders are defined by same-area (Web) companies or same time interval projects. Different companies and different time intervals may mean different geographical locations, possibly different development languages or development methodologies. Such single features are deemed to define cross data borders that hinder knowledge transfer. However, the selection tendencies of the test instances are in disagreement with defining cross borders according to single features. Given the option, test instances select training instances equally likely from within and cross data sources by using all the features of a data set via instance transfer methods like TEAK.

9.7 Conclusion

In this chapter we questioned two fundamental problems of cross data usage: 1) Cross data performance and 2) selection tendency. We challenged previous findings [2, 19, 22, 23, 64] via transfer learning experiments on contemporary projects coming from Web development companies and projects from different time periods. Our findings are in agreement with the prior results. Regarding cross data performance between domains, our analysis showed that cross data performance is indistinguishable from within data performance for majority (5 or 6 out of 8, depending on the error measure) of the data sets. However, a minority of companies (2 out of 8) are better off using their own within data. Practitioners should also be warned that for some cases, different error measures lead to different conclusions. Our recommendations to practitioners regarding the cases of conflicting error measures are:

- to use a number of different error measures;
- and to make their decisions based on:
  - either the agreement of the majority of the error measures
  - or the particular error measure favoring their priorities.

The selection tendency results showed that the definitions of cross and within data based on single features like company name or time interval may be handled with instance transfer methods. Our results show that test instances select both from within and cross data sources. The meaning of this result is that the most similar project(s) to the one currently being estimated is not necessarily within the same company or time interval, but it may be in a cross data set collected on the other side of the world in another time period. This research shows that instance transfer methods like TEAK can make it possible for companies to automatically prune away irrelevant projects and transfer knowledge from relevant training data.

As a consequence of the transfer learning research presented in this chapter, we recommend the following principle regarding the use cross company data for their local data issues:

Principle #9: Use Relevancy Filtering

In summary, the contributions of the research that led to the above principle are:

- Utilization of a novel method for time interval transfer.
- Evaluation of the proposed transfer learning method on recent proprietary as well as public data sets; hence, providing evidence for practitioners as well as benchmark availability for further research.
- To evaluate the previous cross-company research in SE from a transfer learning perspective.

Chapter 10

Principle #10, Use Outlier Pruning

Although SEE is a maturing field of SE with hundreds of publications, among which a significant portion (61% according to a recent survey by Jorgensen and Shepperd [3]) offers new estimation methods, the number of companies interested in SEE is limited. For example, commercial companies like Google[1] *and Microsoft*[2] *and others (e.g. from Turkey [16]) do not use an algorithmic estimation method. The lack of adoption of methods suggested by SEE research in industrial environments is a serious issue. Without addressing the problems that hinder the knowledge transfer from SEE research to industry, the future of SEE is bound to be a mostly theoretical field rather than a practical one. In this chapter, we target this very problem. We will introduce a method called pop1NN that compensates for the removal of the size attributes by removing the outliers (i.e. non-popular instances without neighbors) before the estimation.*

There are various reasons from which the lack of adoption of SEE methods stems, such as the difficulty of attaining accurate estimates, difficulty of collecting effort related data and the difficulty of adopting complex methods. It is possible to increase the items in this list. However, at the heart of the most widely accepted SEE methods lies the measurement of software size. For example, parametric models such as COCOMO use LOC to measure software size and FP approaches use the number of basic logical transactions in a software system. Accurate measurement of both LOC and FP can be problematic in industrial settings [16, 142].

Aside from the difficulty of accurate measurement, the concept of measuring the *"size"* of software is not well adopted. Quoting the former CEO of Microsoft, Bill Gates [142]:*"Measuring software productivity by lines of code is like measuring progress on an airplane by how much it*

[1]Personal communication.
[2]Personal communication.

weighs." A similar notion is also adopted by Dijkstra [143]: *"... This (referring to measuring productivity through LOC) is a very costly measuring unit because it encourages the writing of insipid code, but today I am less interested in how foolish a unit it is from even a pure business point of view. "*

Our notion that we will present in this chapter is that it is possible to develop SEE methods that avoid the use of software size. We propose an estimation method that works without size features (through pruning of non-popular instances, i.e. outliers), yet can attain performance values as good as methods that use size features. Hence, we recommend the *"use outlier pruning"* principle in order to compensate for the lack of size features in SEE. We see the implications of this principle used for the compensation of the lack of size attributes as threefold:

- Promoting development of SEE methods that do not require software size features.
- A proof-of-concept for data collection activities (in research and in industry) that size is not a "must".
- Providing industry practitioners an easy-to-adopt estimation method that does not require size features.

10.1 Notes on the Datasets of This Chapter

In this chapter we will make use of the Cocomo81, Nasa93, Desharnais data sets (as well as their subsets) and SDR. We start with the definition of two keywords that will be fundamental to our discussion in this chapter: *full data set* and *reduced data set*. We will refer to a data set used with all the features (including the size feature(s)) as a *"full data set."* A data set whose size feature(s) are removed will be called a *"reduced data set."* The features of the full data sets are given in Table 10.1. Note in Table 10.1 that the full data sets are grouped under 3 categories (under the *"Methodology"* column) depending on their collection method: COCOMO [20], COCOMOII [144] and FP [30, 145]. These groupings mean that all the data sets in one group share the features listed under the *"Features"* column. The difference between COCOMO data sets (cocomo81* and nasa93*) and COCOMOII data sets (sdr) is the additional five cost drivers: *prec, flex, resl, team, pmat.* Hence instead of repeating the COCOMO features for COCOMOII, we listed

Table 10.1: The full data sets, their features and collection methodology. The bold-face features are identified as size (or size related) features.

Methodology	**Dataset**	**Features**
COCOMO	**cocomo81**	RELY, ACAP, SCED,
	cocomo81o	DATA, AEXP, **KLOC**,
	cocomo81s	CPLX, PCAP, EFFORT,
	nasa93	TIME, VEXP,
	nasa93c1	STOR, LEXP,
	nasa93c2	VIRT, MODP,
	nasa93c5	TURN, TOOL,
COCOMOII	**sdr**	*addition to COCOMO features*:
		PREC,
		FLEX,
		RESL,
		TEAM,
		PMAT
FP	**desharnais**	TeamExp, Effort, **Adjustment**,
	desharnaisL1	ManagerExp, **Transactions**, **PointsAjust**,
	desharnaisL2	YearEnd, **Entities**, Language,
	desharnaisL3	**PointsNonAdjust**

only the additional cost driver features for COCOMOII under the column *"Features"*.

The bold-font features in Table 10.1 are identified as size (or size related) features. These features are removed in reduced data sets. In other words, the full data sets minus the highlighted features gives us the reduced data sets. For convenience, the features that remain after removing the size features are given in Table 10.2. Note that both in Table 10.1 and in Table 10.2, the acronyms of the features are used. These acronyms stand for various software product related features. For example COCOMO groups features under 6 categories:

- Product Factors
  - RELY: Required Software Reliability
  - DATA: Database Size
  - CPLX: Product Complexity
  - RUSE: Required Reusability
  - DOCU: Documentation match to life-cycle needs
- Platform Factors

  - TIME: Execution Time Constraint
  - STOR: Main Storage Constraint
  - PVOL: Platform Volatility

- Personnel Factors
  - ACAP: Analyst Capability
  - PCAP: Programmer Capability
  - PCON: Personnel Continuity
  - AEXP: Applications Experience
  - PEXP: Platform Experience
  - LTEX: Language and Tool Experience

- Project Factors
  - TOOL: Use of Software Tools
  - SITE: Multi-site Development
  - SCED: Development Schedule

- Input
  - LOC: Lines of Code

- Output
  - EFFORT: Effort spent for project in terms of man month

In addition to the original COCOMO method, the improved COCOMOII version defines an additional new category called exponential cost drivers, under which the following features are defined:

- Exponential Cost Drivers:
  - PREC: Precedentedness
  - FLEX: Development Flexibility
  - RESL: Arch/Risk Resolution

  - TEAM: Team Cohesion
  - PMAT: Process Maturity

For detailed information and an in depth discussion regarding the above-listed COCOMO and COCOMOII features refer to [20, 144]. The FP approach adopts a different strategy than COCOMO. The definitions of the FP data sets (desharnais*) features are as follows:

- *TeamExp*: Team experience in years
- *ManagerExp*: Project management experience in years
- *YearEnd*: The year in which the project ended
- *Transactions*: The count of basic logical transactions
- *Entities*: The number of entities in the systems data model
- *PointsNonAdjust* : Equal to *Transactions* + *Entities*
- *Adjustment*: Function point complexity adjustment factor
- *PointsAdjust*: The adjusted function points
- *Language*: Categorical variable for programming language
- *Effort*: The actual effort measured in person-hours

For more details on these features refer to the work of Desharnais [145] or Li et al. [146]. Note that 4 projects out of 81 in desharnais data set have missing feature values. Instead of removing these projects from the data set, we employed a missing value handling technique called simple mean imputation [45].

10.2 Proposed Method: pop1NN

The method proposed in this chapter is a variant of the 1NN algorithm. The proposed variant makes use of the popularity of the instances in a training set. We define the *"popularity"* of an instance as the number of times it happens to be the nearest-neighbor of other instances. The proposed method is called pop1NN (short for popularity-based-1NN). The basic steps of pop1NN can be defined as follows:

Table 10.2: The reduced data sets, their collection methodology and their non-size features. Reduced data sets are defined to be the full data sets minus size-related features (bold-face features of Table 10.1).

Methodology	**Dataset**	**Features**
COCOMO	**cocomo81**	RELY, ACAP, SCED,
	cocomo81o	DATA, AEXP,
	cocomo81s	CPLX, PCAP, EFFORT,
	nasa93	TIME, VEXP,
	nasa93c1	STOR, LEXP,
	nasa93c2	VIRT, MODP,
	nasa93c5	TURN, TOOL,
COCOMOII	**sdr**	*addition to COCOMO features:*
		PREC,
		FLEX,
		RESL,
		TEAM,
		PMAT
FP	**desharnais**	TeamExp, Effort,
	desharnaisL1	ManagerExp
	desharnaisL2	YearEnd, Language
	desharnaisL3	

Step 1: Calculate distances between every instance tuple in the training set.

Step 2: Convert distances of Step 1 into ordering of neighbors.

Step 3: Mark closest neighbors and calculate popularity.

Step 4: Order training instances in decreasing popularity.

Step 5: Decide which instances to select.

Step 6: Return Estimates for the test instances.

The following paragraphs describe the details of these steps:

Step 1: Calculate distances between every instance tuple in the training set: This step uses the Euclidean distance function (as in 1NN) to calculate the distances between every pair of instances within the training set. The distance calculation is kept in a matrix called D, where i^{th} row keeps the distance of the i^{th} instance to other instances. Note that this calculation requires only the independent features. Furthermore, since pop1NN runs on reduced data sets, size features are not used in this step .

Step 2: Convert distances of Step 1 into ordering of neighbors: This step requires us to merely replace the distance values with their corresponding ranking. We work one row at a time on matrix D: Start from row #1, rank distance values in ascending order, then replace distance values with their corresponding ranks, which gives us the matrix D'. The i^{th} row of D' keeps the ranks of the neighbors of the i^{th} instance.

Step 3: Mark closest neighbors and calculate popularity: Since pop1NN uses only the closest neighbors, we leave the cells of D' that contain 1 (i.e. that contains a closest neighbor) untouched and replace the contents of all the other cells with zeros. The remaining matrix D'' marks only the instances that appeared as the closest neighbor to another instance.

Step 4: Order training instances in decreasing popularity: This step starts summing up the *"columns"* of D''. The sum of, say, i^{th} column shows how many times the i^{th} instance was marked as the closest neighbor to another instance. The sum of the i^{th} column equals the popularity of the i^{th} instance. Finally in this step, we rank the instances in decreasing popularity, i.e. the most popular instance is ranked #1, the second is ranked #2 and so on.

Step 5: Decide which instances to select: This step tries to find how many of the most popular instances will be selected. For that purpose we perform a 10-way cross validation on the train set. For each cross-validation (i.e. 10 times), we do the following:

- Perform steps 1 to 4 for the popularity order;
- Build a set S, into which instances are added one at a time from the most popular to the least popular;
- After each addition to S make predictions for the hold out set, i.e. find the closest neighbor from S of each instance in the hold-out set and use the effort value of that closest instance as the estimate;
- Calculate the error measure of the hold-out set for each size of S. As the size of S increases (i.e. as we place more and more popular instances into S) the error measure is expected to decrease;

- Traverse the error measures of S with only one instance to S with t instances (where t is the size of the training set minus the hold-out set). Mark the size of S (represented by s') when the error measure has not decreased more than Δ for $n - many$ consecutive times.

Note that since we use a 10-way cross-validation, at the end of the above steps, we will have 10 s' values (one s' value from each cross-validation). We take the median of these values as the final s' value. This means that pop1NN only selects the most popular s'-many instances from the training set. For convenience, we refer to the new training set of selected s'-many instances as $Train'$.

Step 6: Return Estimates for the Test Instances: This step is fairly straightforward. The estimate for a test instance is the effort value of its nearest neighbor in $Train'$.

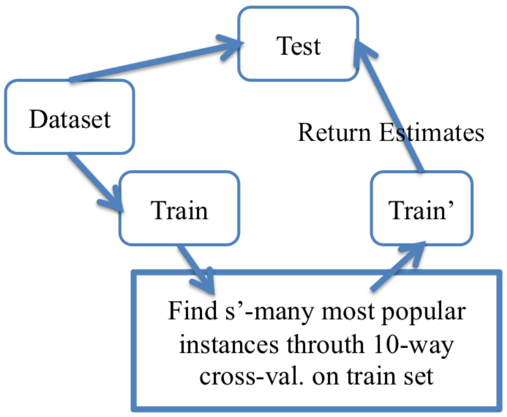

Test
Dataset
Return Estimates
Train
Train'
Find s'-many most popular instances throuth 10-way cross-val. on train set

Figure 10.1: A simple illustration of the pop1NN method. Note that the test and train sets are generated through a 10-way cross-validation as well.

For the error measure in *Step 5*, we used "MRE", which is only one of the many possible error measures. Even though we guide the search using only MRE, the resulting estimations (as shown in the Result Section of this chapter) score very well across a wide range of error measures. In the following experiments, we used $n = 3$ and $\Delta < 0.1$.

10.3 Experiments

The experiments are performed in two stages: 1) 1NN and CART performances on reduced data sets compared to their performance on full data sets; 2) pop1NN performance on reduced data set compared to 1NN and CART performance on full data sets.

In the first stage we question whether standard SEE methods can compensate mere removal of the size features. For that purpose we run 1NN as well as CART on reduced and full data sets separately through 10-way cross-validation. Then each method's results on reduced data sets are compared to its results on full data set. The outcome of this stage tells us whether there is a need for pop1NN like methods or not. If the performance of CART and 1NN on reduced data sets are statistically the same as their performances on the full data sets, then this would mean that standard successful estimation methods are able to compensate the lack of size features. However, as we will see in the Results Section, that is not the case. The removal of size features has a negative effect on 1NN and CART.

The second stage tries to answer whether simple SEE methods like 1NN can be augmented with a pre-processing step, so that the removal of size features can be tolerated. For that purpose we run pop1NN on the reduced data sets and compare its performance to 1NN and CART (run on full data sets) through 10 way cross-validation. The performance is measured in 7 error measures: MAR, MMER, MMRE, MdMRE, Pred(25), MBRE and MIBRE.

10.4 Results

10.4.1 Results Without Instance Selection

Table 10.3 shows the CART results for the first stage of our experimentation, i.e. whether or not standard estimation methods, in that case CART, can compensate for the lack of size. In Table 10.3 we compare CART run on reduced data sets to CART run on full data sets and report the loss values. The loss value in each cell is associated with an error measure and a data set. Each loss value shows whether CART on reduced data lost against CART on full data. Note that it is acceptable for CART-on-reduced-data, as long as it does not lose against CART-on-full-data, since we want the former to perform just as well as (not necessarily better than) the latter. The last column of Table 10.3 is the sum of the loss values over 7 error measures. The rows in which CART on reduced data loses for most of the error measures (4 or more out of 7 error measures) are highlighted. See in Table 10.3 that 7 out of 13 data sets are highlighted, i.e. more than half the time CART cannot compensate the lack of size features.

Although Table 10.3 is good to see the detailed loss information, the fundamental information we are after is summarized in the last column: total loss number. Repeating Table 10.3 for all the methods in both stages of the experimentation is cumbersome and would redundantly take too much space. Hence, from now on we will use summary tables as given in Table 10.4, which shows only the total number of losses. See that the *"CART"* column of Table 10.4 is just the last column of Table 10.3 . Aside from the CART results, Table 10.4 also shows the loss results for 1NN run on reduced data vs. 1NN run on full data. The highlighted cells of *"1NN"* column show the cases, where 1NN lost most of the time, i.e. 4 or more out of 7 error measures. Similar to the results of CART, 1NN-on-reduced-data loses against 1NN-on-full-data for 7 out of 13 data sets. In other words, for more than half the data sets mere use of 1NN cannot compensate for the lack of size features.

The summary of the first stage of experimentation is that standard SEE methods are unable to compensate for the size features for most of the data sets. In the next section we show the interesting result that it is possible to augment a very simple ABE method like 1NN so that it can compensate for size features in a big majority of the data sets.

Table 10.3: The loss values of CART run on reduced data set vs. CART run on full data set, measured per error measure. The last column shows the loss values in total of 7 error measures. The data sets where CART running on reduced data sets lose more than half the time (i.e. 4 or more out of 7 error measures) against CART running on full data sets are highlighted.

	MMRE	**MAR**	**Pred(25)**	**MdMRE**	**MBRE**	**MIBRE**	**MMER**	**Total**
cocomo81	1	0	1	1	1	1	0	5
cocomo81e	0	0	0	0	0	0	0	0
cocomo81o	0	0	0	0	0	0	0	0
cocomo81s	0	0	0	0	0	0	0	0
desharnais	1	0	1	1	1	1	0	5
desharnaisL1	1	1	1	1	1	1	1	7
desharnaisL2	0	0	0	0	0	0	0	0
desharnaisL3	0	0	0	0	0	0	0	0
nasa93	1	1	1	1	1	1	1	7
nasa93c1	1	0	1	1	1	1	0	5
nasa93c2	1	1	1	1	1	1	1	7
nasa93c5	0	0	0	0	0	0	0	0
sdr	1	0	1	1	0	0	1	4

Table 10.4: The loss values of estimation methods run on reduced data sets vs. run on full data sets. The cases where reduced data set results lose more than half the time (i.e. 4 or more out of 7 error measures) are highlighted.

		Methods	
		CART	**1NN**
Data Sets	**cocomo81**	5	5
	cocomo81e	0	0
	cocomo81o	0	0
	cocomo81s	0	0
	desharnais	5	7
	desharnaisL1	7	7
	desharnaisL2	0	2
	desharnaisL3	0	0
	nasa93	7	7
	nasa93c1	5	6
	nasa93c2	7	7
	nasa93c5	0	6
	sdr	4	0

10.4.2 Results With Instance Selection

The comparison of pop1NN (which runs on reduced data sets) to 1NN and CART (which run on full data sets) is given in Table 10.5. Table 10.5 shows the total loss values of pop1NN over 7 error measures, so the highest number of times pop1NN can lose against 1NN or CART is 7. The cases where pop1NN loses more than half the time (i.e. 4 or more out of 7 error measures) are highlighted.

The comparison of pop1NN against 1NN shows, whether the proposed pop1NN method helps standard ABE methods to compensate for the lack of size features. See that in the second column of Table 10.5 there are only 2 highlighted cells. For 11 out of 13 data sets, the performance of pop1NN is statistically the same to that of 1NN. The only 2 data sets, where pop1NN cannot compensate for size are the *cocomo81e* and *desharnais*.

The fact that pop1NN compensates for size in cocomo81e's superset (cocomo81) and in desharnis' subsets (desharnaisL1, desharnaisL2 and desharnaisL3) but not in these two data sets may at first seem puzzling. Because, the expectation is that subsets share similar properties as their supersets. However, a recent work by Posnett et al. have shown that this is not necessarily the case [147]. The focus of Posnet et al.'s work is the "ecological inference"; i.e. the delta between the conclusions drawn from subsets vs. the conclusions from the supersets. They document the interesting finding that conclusions from the subsets may be significantly different to the conclusions drawn from the supersets. Our results support their claim that supersets and subsets may have different characteristics.

The last column of Table 10.5 shows the number of times pop1NN lost against CART. Again the cases where pop1NN loses for more than 4 error measures are highlighted. The purpose of pop1NN's comparison to CART is to evaluate a simple ABE method like pop1NN against a state-of-the-art learner like CART. As can be seen in Table 10.5, there are 4 highlighted cells under the last column, i.e. for *13-4=9* data sets, the performance of pop1NN is statistically same to that of CART. This is an important result for two reasons: 1) a simple ABE method like pop1NN can attain performance values as good as CART for most of the data sets; 2) the performance of pop1NN comes from data sets without any size features.

10.5 Discussion

An important point of discussion is the meaning of our results for practitioners. Should the size features and the models built on size features be abandoned? The answer is simply: "No." Parametric methods whose fundamental input is size, like COCOMO, can be calibrated to local environments for high estimation performances. Also in the absence of size features, machine

Table 10.5: The loss values of pop1NN vs. 1NN and CART over 7 error measures. The data sets where pop1NN (running on reduced data sets) lose more than half the time (i.e. 4 or more out of 7 error measures) against 1NN or CART (running on full data sets) are highlighted.

	pop1NN vs. 1NN	**pop1NN vs. CART**
cocomo81	0	3
cocomo81e	7	3
cocomo81o	0	7
cocomo81s	0	0
desharnais	7	5
desharnaisL1	0	0
desharnaisL2	0	7
desharnaisL3	0	0
nasa93	3	0
nasa93c1	0	7
nasa93c2	0	3
nasa93c5	0	0
sdr	0	0

learning methods such as 1NN and CART perform poorly. Hence, it would be a misinterpretation of this study to claim that size features are deprecated. On the other hand, if practitioners need simple methods for the cases, where measuring size features accurately (or at all) is not possible, then: 1) the use of parametric methods may be questionable and 2) the use of machine learning methods (e.g. 1NN and CART) may yield low estimation performances. For such cases the use of methods like pop1NN can actually compensate for the lack of size features and provide an alternative solution to practitioners.

10.6 Conclusion

Size features are fundamental to many estimation methods such as COCOMO, COCOMOII, FP and so on. In this chapter we question whether the size features are indispensable or not. We evaluate 1NN and CART, which are reported as the best methods out of 90 methods in a prior study [148]. Our results show that the performance of 1NN and CART on reduced data sets are worse than their performance on full data sets. Hence, mere use of these methods without size features is not recommended.

Then we augmented 1NN with a popularity based pre-processor to come up with pop1NN. We run pop1NN on reduced data sets and compare its performance to 1NN and CART, which are both run on full data sets. The results of this comparison show that for most of the cases (11 out of 13 data sets), pop1NN running on reduced data sets attains the same performance as its counterpart 1NN running on full data sets. Hence, pop1NN can compensate for the lack of size features for a big majority of the cases. Also, for 9 out of 14 datasets, a Euclidean distance based learner like pop1NN even attains performance values that are statistically significantly the same as a state-of-the-art learner like CART.

Size features are essential for standard learners such as 1NN and CART. SEE practitioners with enough resources to collect accurate size features should do so. On the other hand, when standard learners (in this research it is 1NN) are augmented with pre-processing options, it is possible to remove the necessity of size features. Hence, SEE practitioners without sufficient resources to measure accurate size features should consider alternatives like pop1NN and the recommended principle as a result of this chapter is:

$\boxed{\text{Principle \#10: Use Outlier Pruning}}$

The contributions of the research presented in this chapter can be summarized as follows:

- Promotion of SEE methods that can compensate the lack of the software size features.
- A method called pop1NN that shows that size features are not a “must”.

Chapter 11

Principle #11, Combine Outlier and Synonym Pruning

In the previous chapters of this thesis, we have seen that there is a wide variety of estimation methods used in SEE. Particularly in Chapter 5, we have evaluated a large number of methods of varying complexity. In SEE, the justification for the complexity of the introduced method is a critical yet mostly overlooked issue. In this chapter we will argue that the complexity of the learning method should be matched the to essential content *of the data.*

Given a matrix of N instances and F features, the essential content is $N' * F'$ where N' and F' are subsets of the instances and features, respectively. The models learned from N' and F' perform as well as the models learned from N and F. In this chapter, we are interested in discovering N' and F' using a novel method called QUICK, a tool that is remarkably simple. QUICK computes the Euclidean distance between the rows (instances) of SEE data sets. That distance calculation is also performed between matrix columns (features) using a transposed copy of the matrix. QUICK then removes synonyms (features that are very close to other features) and outliers (rows that are very distant to the other rows). QUICK then reuses the distance calculations a final time to find estimates for test cases, using their nearest neighbor in the reduced space. Note that QUICK is an improvement over the pop1NN algorithm, which was introduced in the previous chapter. Recall that in the previous chapter pop1NN used popularity of the instances in order to remove the size requirements in SEE data sets. QUICK builds on to pop1NN and in addition to the removal of the outliers, it also performs synonym pruning. As we will see later on in this chapter, synonyms are the features that are nearest neighbors to a high number of other features. The purpose of QUICK in this chapter is to discover the essential content of SEE data sets and to maintain low error rates.

We show in this chapter that QUICK's estimates from the reduced space are as good as those of a state-of-the-art method (that uses all the data). Further, it does so using only 10% (or less) of the original data- see Table 11.1.

Our only explanation for the success of such a fundamentally simple method (using such tiny regions of the original data) is that the essential content of effort data sets *can be localized in just a few rows and just a few columns*. If so, then:

- Elaborate estimation methods will learn little more than simpler ones (since there is so little to find).
- We can recommend simpler methods (e.g. QUICK).

11.1 Why the Need for Simpler Methods?

In this chapter we show that QUICK is able to reduce the SEE data sets to a fraction of the instances and the features and we support the principle that states "Combine Outlier and Synonym Pruning". We argue that it is possible to reduce the complexity of the SEE methods. But why should we bother? If the complex methods already exist, and they work, why explore simpler ones? We have five replies to this challenge.

11.1.1 Elaborate on the Data, not on the Algorithms

We are concerned about excessive over-elaboration of the effort estimation problem. Since the essential content of effort data sets is so small, it may be unproductive to explore further elaborations of effort estimation methods. Rather, it may be more productive to increase the data's information density. Although it is not in the scope of our research presented in this chapter, we want to note that this is an active area of research. For example, researchers have proposed:

- Combining data sets from different sources [2];
- Adding contextual knowledge to represent more business information, as explored by many including Petersen & Wohlin [149];

Table 11.1: Last column shows fraction of data $\frac{N'*F'}{N*F}$ selected by QUICK.

Dataset	**Used by**	**% Selected Cells**
nasa93	[152–154]	4%
cocomo81	[20, 153, 154]	7%
cocomo81o	[122, 155, 156]	7%
maxwell	[154, 157, 158]	8%
kemerer	[38, 154, 159]	9%
desharnais	[105, 120, 157]	9%
miyazaki94	[37]	10%
desharnaisL1	[46]	10%
nasa93_center_5	[122, 160, 161]	11%
nasa93_center_2	[122, 160, 161]	11%
cocomo81e	[32, 156, 160]	14%
desharnaisL2	[19]	14%
sdr	[90, 154, 162]	16%
desharnaisL3	[19]	18%
nasa93_center_1	[122, 160, 161]	19%
cocomo81s	[122, 156]	20%
finnish	[38, 111]	26%
albrecht	[38, 46, 157]	31%

- Dividing existing data into subsets more relevant to particular business units, as explored by ourselves and Posnett et al. [147, 150];

- Understanding how the relevant parts of a data set changes as we move from older projects to newer projects, as explored by Passos et al. [151].

One of our aims with the research presented here is to encourage more work on increasing information density, while discouraging research on needlessly elaborate effort estimation models.

11.1.2 Simpler Explanation and Wider Adoption

Business users often need justification of how an estimate was reached. Explaining (e.g.) the 100 methods used in our ensemble learner [148] is a non-trivial matter. On the other hand, explaining QUICK is very simple:

- Group rows and columns by their similarity;

- Discard redundant columns that are too similar;

- Discard outlier rows that are too distant;

Table 11.2: Explanation of the symbols that will be used in the rest of this chapter.

Symbol	Meaning
D	Denotes a specific data set
D'	Denotes the reduced version of D by QUICK
D^T	Transposed version of D
N	Number of instances in D
N'	Number of *selected* instances by QUICK from D
F	Number of independent features in D
F'	Number of independent features by QUICK from D
P	Represents a project in D
$Feat$	Represents a feature in D
i, j	Subscripts used for enumeration
P_i, P_j	i^{th} and j^{th} projects of D, respectively
DM	An $N \times N$ distance matrix that keeps distances between all projects of D
$DM(i, j)$	The distance value between P_i and P_j
k	Number of analogies used for estimation
E_{NN}	Everyone's nearest-neighbor matrix, which shows the order of P's neighbors w.r.t. a distance measure
$E(k)$	$E(k)[i, j]$ is 1 for $i \neq j$ and $E_{NN} \leq k$, 0 otherwise
$E(1)$	Specific case of $E(k)$, where $E(1)[i, j]$ is 1 if j is i's closest neighbor, 0 otherwise
$E(1)[i, j]$	The cell of $E(1)$ that corresponds to i^{th} row and j^{th} column
$Pop(j)$	Popularity of j: $Pop(j) = \sum_{i=1}^{n} E(1)[i, j]$
n	Number of projects consecutively added to active pool, i.e. a subset of D, i.e. $n \leq N$
Δ	The difference between the best and the worst error of the last n instances.
$x_i, \hat{x}_i$	Actual and predicted effort values of P_i, respectively.
$Perf_i, Perf_i$	i^{th} and j^{th} performance measures
M_i, M_i	i^{th} and j^{th} estimation methods

- In the remaining data, generate an estimate from the nearest example.

Such simple explanations increase the acceptance of an estimation method, amongst senior business users as well as more junior engineers. For example, when engineers implement methods like COSEEKMO [26] or ensemble methods [148], they complain to us about the complexity of those methods (those tools require the implementation and execution of hundreds of variants of different estimation models). While we have shown that COSEEKMO and the ensemble method are effective, it is still a legitimate engineering question to ask "how much can we achieve with far fewer methods?".

11.1.3 More Complexity = More Operator Error

The more complex the methods become, the more prone they become to operator error. This is a growing problem. Shepperd et al. [163] report that the dominant factor that predicts for model performance is *who operates the data miner* (and not which data set is studied; and not which data miner is used). This is a very troubling result that suggests our sophisticated data mining methods are now so complex, that they have become very troublesome and error-prone. One of the main points of this chapter is that the complexity inherent in an SEE method is only necessary, if the value-added is worth it. The experiments presented in the rest of this chapter show that for SEE, the value-added above a very simple set of nearest neighbor calculations (i.e. those used in QUICK) is minimal at best.

11.1.4 A New Ability to Certify Simpler Solutions

For the first time in the history of this field, it is possible to conduct simplification studies. In the last year, there have been published two studies in IEEE TSE commenting on what effort estimation methods are best [148, 164]. These studies assert that, in terms of assessing new effort estimation methods, existing methods such as CART's regression tree generation may be more than adequate. For example, Dejaeger et al. found little evidence that learners more elaborate than CART offered any significant value-added [164]. Also, in our study with 90 effort estimators that use all combinations of 10 pre-processors and 9 learners to build ensemble methods [148], we found that CART was all-round best performer. Hence, in the following, we will compare QUICK (running on the reduced data) with CART (running on all the data).

11.2 About QUICK

11.2.1 Pruning Synonyms

Synonyms are features that are closely associated to each other. QUICK detects and removes these redundant features as follows.

1) Take an unlabeled data set (i.e. no effort value) and transpose it, s.t. the independent features become the instances (rows) in a space whose dimensions (columns) are defined by the original

instances.

2) Let E_{NN} be a matrix ranking the neighbors of a feature: closest feature is $\#1$, the next is $\#2$ and so on.

3) $E(k)$ marks *k*-closest features (e.g. *E(1)* marks $\#1$'s).

4) Calculate the *popularity* of a feature by summing up how many times it was marked as $\#1$.

5) Leave only the unmarked features, i.e. the ones with a popularity of zero. These features are not closest neighbor to others.

11.2.2 Pruning Outliers

Outlier pruning uses the same machinery as synonym pruning- with certain important differences:

- With synonym pruning, we *transpose* the data to find the distances between "rows" (which, in the transposed data, are really features). Then we count the *popularity* of each feature and delete the *popular* features (these are the features that needlessly repeated the information found in other features).

- With outlier pruning, we *do not transpose* the data before finding the distances between rows. Then we count the *popularity* of each row and delete the *unpopular* rows (these are the instances that are most distant to the others).

Note that the data set used to prune outliers contains only the selected features of the previous phase. Also note that the terms feature and variable will be used interchangeably from now on. Following is the brief description of the outlier pruning phase:

1) Let E_{NN} be a matrix ranking the neighbors of a project, i.e. closest project is $\#1$, the next is $\#2$ etc.

2) $E(k)$ marks only the *k*-closest projects, e.g. *E(1)* marks only $\#1$'s.

3) Calculate the *popularity* of a project by summing up how many times it was marked as $\#1$.

4) Collect the costing data, one project at a time in decreasing *popularity*, then place it in the "active pool".

5) Before a project's, say project A, costing data is placed into the active pool, generate an estimate

for A. The estimate is the costing data of A's closest neighbor in the active pool. Note the difference between the estimate and the actual cost of A.

6) Stop collecting costing data if the active pool's error rate does not decrease in 3 consecutive project additions.

7) Once the stopping point is found, a future project -say project B- can be estimated. QUICK's estimate is the costing data of B's closest neighbor in the active pool.

11.2.3 More Details

The above paragraphs are informal descriptions, which are repeated, in full detail in §11.5. Note that we make no claim for optimality- we do not presume that QUICK finds the absolute smallest set of features and rows. As long as we can show that the selected subset of the data is very small and that simple estimation methods executing in this reduced space perform as well as state-of-the-art methods executing over all the data, then the goal of the research presented in this chapter is attained.

11.3 Active Learning

One way to view QUICK's outlier pruning method is an *active learning* method. In active learning, some heuristic is used to sort instances from most interesting to least interesting (in our case, that heuristic is each row's *popularity* value). The data is then explored according to that sort order. Learning can terminate early if the results from all the N instances are not better than from a subset of M instances, where $M < N$.

There is a wealth of active learning studies in machine learning literature. For example Dasgupta et al. [165] seek for generalizability guarantees in active learning. They use a greedy active learning heuristic and show that it is able to deliver performance values as good as any other heuristic, in terms of reducing the number of required labels [165]. In [166], Wallace et al. used active learning for a deployed practical application example. They propose a citation screening model based on active learning augmented with a priori expert knowledge.

In software engineering, the practical application exemplars of active learning can be found in software testing [167, 168]. In Bowring et al.'s study [167], active learning is used to augment learners for automatic classification of program behavior. Xie et al. [168] use human inspection as an active learning strategy for effective test generation and specification inference. In their experiments, the number of tests selected for human inspection were feasible; the direct implication is that labeling required significantly less effort than screening all single test cases. Hassan et al. list active learning as part of the future of software engineering data mining [169]. However, to the best of our knowledge this promising direction for data analysis has not been explored in SEE.

11.4 Feature Subset Selection

Another way to view QUICK's synonym pruning is as an *unsupervised feature subset selection* (FSS) algorithm. It is well known that selecting a subset of the SEE data set features can improve the estimation performance. For example Chen et al. use WRAPPER FSS algorithm on COCOMO data sets and report dramatic increases in the estimation performance [170]. Azzeh et al. propose a fuzzy feature subset selection method in analogy-based SEE [171]. They report improvements over using all features as well as over standard FSS methods such as forward and backward feature subset selection. Lum et al. capture and report the best practices in SEE [122]. One of the fundamental suggestions among the best practices is FSS. They show that both manual and supervised FSS methods improve estimation performance [122]. Other FSS examples are the stepwise regression (SWR) [111] and principal component analysis (PCA) [148, 172].

It is possible to find many other studies supporting the adoption of FSS in SEE. For example Kadoda et al. propose FSS as a performance improvement technique for analogy-based estimation [173]. In a similar manner Kirsopp and Shepperd show that FSS improves estimation performance in case-based reasoning [120]. The common property of the aforementioned FSS algorithms except PCA is that they are *supervised*. *Supervised algorithms* require the instance labels (i.e. dependent variables). QUICK on the other hand is an *unsupervised* FSS algorithm. Unlike supervised algorithms, QUICK can execute without labels, which removes the necessity of label collection prior to FSS. In comparison to PCA QUICK's method of finding feature subsets is easier to implement and understand, in particular for those without machine learning background. QUICK only requires the knowledge of normalization of an array of instances and the calculation of the Euclidean distance. PCA on the other hand requires the user to know the concepts of correlation between the features as well as the orthogonal transformation [45]. Furthermore, unlike QUICK, PCA's output is not a subset of the individual features that user sees on the data sets, but rather a new set of -less correlated- features (the principal components), which are linear combinations of the original features.

11.5 The Application of QUICK

QUICK is an active learner that works in two phases:

1. Synonym pruning (a.k.a. feature selection);
2. Outlier removal (a.k.a. instance selection) and estimation.

Each phase requires a number of execution steps, some of which are shared. The following paragraphs are the detailed descriptions of the steps in each phase.

11.5.1 Phase1: Synonym Pruning

1. Transpose data set matrix;
2. Generate distance-matrices;
3. Generate $E(k)$ matrices using E_{NN};
4. Calculate the popularity index based on $E(1)$ and select non-popular features.

1. Transpose data set matrix: This step may or may not be necessary depending on how the initial data set is stored. However, rows of SEE data sets usually represent the past project instances, whereas the columns represent the features defining these projects. When such a matrix is transposed the project instances are represented by columns and project features are represented by the rows. Note that columns are normalized to 0-1 interval before transposing to remove the superfluous effect of large numbers in the next step.

2. Generate distance-matrices: For the transposed dataset D^T of size F, the associated distance-matrix (DM) is an $F \times F$ matrix keeping the distances between every feature-pair according to Euclidean distance function. For example, a cell located at i^{th} row and j^{th} column $(DM(i,j))$ keeps the distance between i^{th} and j^{th} features (diagonal cells $(DM(i,i))$ are ignored).

3. Generate E_{NN} and $E(1)$ matrices: $E_{NN}[i,j]$ shows the neighbor rank of "j" w.r.t. "i", e.g. if "j" is "i's" 3^{rd} nearest neighbor, then $E_{NN}[i,j]$=3. The trivial case where i=j is ignored, i.e. an instance's nearest neighbor does not include itself. The $E(k)$ matrix is defined as follows: if $i \neq j$ and $E_{NN}[i,j] \leq k$, then $E(k)[i,j] = 1$; otherwise $E(k)[i,j] = 0$. In synonym pruning, we want to select the unique features without any nearest-neighbors. For that purpose we start with *k*=1; hence E(1) identifies the features that have at least another nearest-neighbor and the ones without any nearest-neighbor. The features that appear as one of the k-closest neighbors of another feature are said to be popular. The "popularity index" (or simply "popularity") of feature "j', $Pop(Feat_j)$,' is defined to be $Pop(Feat_j) = \sum_{i=1}^{n} E(1)[i,j]$, i.e. how often "$j^{th}$" feature is some other feature's nearest-neighbor.

4. Calculate the popularity index based on $E(1)$ and select non-popular features: Non-popular features are the ones that have a popularity of zero, i.e. $Pop(Feat_i) = 0$.

11.5.2 Phase2: Outlier Removal and Estimation

1. Generate distance-matrices;
2. Generate $E(k)$ matrices using E_{NN};
3. Calculate a "popularity" index;
4. After sorting by popularity, find the stopping point.

1. Generate distance-matrices: For a dataset D of size N, the associated distance-matrix (DM) is an $N \times N$ matrix whose cell located at row i and column j $(DM(i,j))$ keeps the distance between i^{th} and j^{th} instances of D. The cells on the diagonal $(DM(i,i))$ are ignored. Note that D in this phase comes from the prior phase of synonym pruning, hence it only has the selected features.

2. Generate E_{NN} and $E(1)$ matrices: $E_{NN}[i,j]$ shows the neighbor rank of "j" w.r.t. "i". Similar to the step of synonym pruning, if "j" is "i's" 3^{rd} nearest neighbor, then $E_{NN}[i,j]$=3. Again the trivial case of i=j is ignored (nearest neighbor does not include itself). The $E(k)$ matrix has exactly the same definition as the one in synonym pruning phase: if $i \neq j$ and $E_{NN}[i,j] \leq k$,

Table 11.3: The percentage of the popular instances (to data set size) useful for prediction in a closest-neighbor setting. Only the instances that are closest neighbors of another instance are said to be popular. The median percentage value is 63%, i.e. $1/3$ of the instances are not the closest neighbors and will never be used by, say, a nearest neighbor system.

Dataset	**% popular instances**
kemerer	80
telecom1	78
nasa93_center_1	75
cocomo81s	73
finnish	71
cocomo81e	68
cocomo81	65
nasa93_center_2	65
nasa93_center_5	64
nasa93	63
cocomo81o	63
sdr	63
desharnaisL1	61
desharnaisL2	60
desharnaisL3	60
miyazaki94	58
desharnais	57
albrecht	54
maxwell	13

then $E(k)[i, j] = 1$; otherwise $E(k)[i, j] = 0$. In this study the nearest-neighbor based ABE is considered, i.e. we use *k*=1; hence E(1) describes just the single nearest-neighbor. All instances that appear as one of the k-closest neighbors of another instance are defined to be popular. The "popularity index" (or simply "popularity") of instance "j', $Pop(j)$,' is defined to be $Pop(j) = \sum_{i=1}^{n} E(1)[i, j]$, i.e. how often "j" is someone else's nearest-neighbor.

3. Calculate the popularity index based on $E(1)$ and determine the sort order for labeling:

As shown in Table 11.3, the popular instances j with $Pop(j) \geq 1$ (equivalently, $E(1)[i, j]$ is 1 for some i) have a median percentage of 63% among all datasets; i.e. more than $1/3$ of the data is unpopular with $Pop(j) = 0$. Following this observation, we speculated that if we label data in order of its popularity, then we would be labeling the most important projects *first*.

4. Find Stopping Point and Halt: The notion behind instance selection is to reach conclusions using fewer instances. To test that, QUICK labels some instances (adding them to the active pool) then stops, we have defined the following stopping rules:

1. All instances with $Pop(j) \geq 1$ are exhausted.
2. Or if there is no estimation accuracy improvement in the active pool for n consecutive times.
3. Or if the Δ between the best and the worst error of the last n instances in the active pool is very small.

For the error measure in point #3, we used "MRE"; i.e. the magnitude of relative error ($abs(actual-predicted)/actual$). MRE is only one of many possible error measures. As shown below, even though we guide the search using only MRE, the resulting estimations score very well across a wide range of error measures.

Note that retaining policy of instances is different to that of features. Instances with high popularity are retained, whereas for the features high popularity is a reason to be discarded. Distant instances without any neighbors are likely to be outliers and we expect an essential set of instances with high popularity to capture the essential content of a data set. Distant features with low popularity are expected to reflect a different view of the data, whereas features with high popularity may be repeating information that could have been conveyed by their neighboring features.

In the following experiments, we used $n = 3$ and $\Delta < 0.1$. The selection of (n, Δ) values is based on our engineering judgment. The sensitivity analysis of trying different values of (n, Δ) can be a promising future work.

11.5.3 Examples

This section offers a small example of QUICK. Assume that the training set of the example consists of 3 instances/projects: P_1, P_2 and P_3. Also assume that these projects have 1 dependent and 3 independent features. Our dataset would look like Figure 11.1.

Project	$Feat_1$	$Feat_2$	$Feat_3$	**Effort**
P_1	1	2	20	3
P_2	2	3	10	4
P_3	3	6	40	7

Figure 11.1: Three projects defined by 3 independent features/variables and a dependent variable (staff-months).

11.5.4 Synonym Pruning

Step 1: *Transpose data set matrix:* After normalization to 0-1 interval, then transposing our data set, the resulting matrix would look like Figure 11.2.

	P_1	P_2	P_3
$Feat_1$	0.0	0.5	1
$Feat_2$	0.0	0.5	1
$Feat_3$	0.3	0.0	1

Figure 11.2: Resulting matrix after normalizing and transposing D.

Step 2: *Generate distance-matrices:* The distance matrix DM keeps the Euclidean distance between features. The matrix of Figure 11.2 is used to calculate the DM of Figure 11.3.

	$Feat_1$	$Feat_2$	$Feat_3$
$Feat_1$	na	0.0	0.6
$Feat_2$	0.0	na	0.6
$Feat_3$	0.6	0.6	na

Figure 11.3: DM for features.

Step 3: *Generate E_{NN} and $E(1)$ matrices:* According to the distance matrix of Figure 11.3 the resulting $E_{NN}[i, j]$ of Figure 11.4 shows the neighbor ranks of features. Using E_{NN} we calculate the E(1) matrix that identifies the features that have at least another nearest-neighbor. E(1) matrix is given in Figure 11.5.

	$Feat_1$	$Feat_2$	$Feat_3$
$Feat_1$	na	1	2
$Feat_2$	1	na	2
$Feat_3$	1	1	na

Figure 11.4: The E_{NN} matrix for features, resulting from the distance matrix of Figure 11.3. Diagonal cells are ignored.

Step 4: *Calculate the popularity index based on $E(1)$ and select non-popular features:* Popularity of a feature is the total of $E(1)$'s columns (see the summation in Figure 11.5). *Non-popular* features are the ones with zero popularity. In this toy example, we only select $Feat_3$, since it is the only column with zero popularity.

		$Feat_1$	$Feat_2$	$Feat_3$
	$Feat_1$	0	1	0
	$Feat_2$	1	0	0
$+$	$Feat_3$	1	1	0
	$Popularity:$	2	1	0

Figure 11.5: Popularity of the features. Popularity is the sum of the $E(1)$ matrix columns.

11.5.5 Outlier Removal and Estimation

In this phase QUICK continues execution with only the selected features. After the first phase, our data set now looks like Figure 11.6.

Project	$Feat_3$	**Effort**
P_1	20	3
P_2	10	4
P_3	40	7

Figure 11.6: Three projects defined by $Feat_3$ (say KLOC) and effort (say, in staff-months).

Since our data now has only 1 independent variable, we can visualize it on a linear scale as in Figure 11.7.

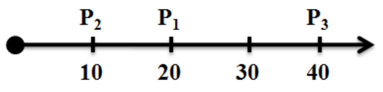


Figure 11.7: Visualization of projects on a linear KLOC scale.

Step 1: The first step of QUICK in this phase is to *build the distance matrix*. Since projects are described by a single attribute $Feat_3$ (say KLOC), the Euclidean distance between two projects will be the difference between the normalized KLOC values. The resulting *distance-matrix* is given in Figure 11.8.

	P_1	P_2	P_3
P_1	0	0.3	0.7
P_2	0.3	0	1
P_3	0.7	1	0

Figure 11.8: The distance matrix of the projects P_1, P_2 and P_3.

Step 2: *Creating the E_{NN} matrix* based on the distance matrix is the second step. As we are creating the E_{NN} matrix we traverse the distance matrix row-by-row and label the instances depending on their distance order: closest neighbor is labeled 1, the second closest neighbor is labeled 2 and so on. Note that diagonal entries with the distance values of 0 are ignored, as they represent the distance of the instance to itself, not to a neighbor. After this traversal, the resulting E_{NN} is given in Figure 11.9.

	P_1	P_2	P_3
P_1	na	1	2
P_2	1	na	2
P_3	1	2	na

Figure 11.9: The E_{NN} matrix resulting from the distance matrix of Figure 11.8. Diagonal cells are ignored.

Step 3: *Calculating the popularity index based on E_{NN} and determining the labeling order* is the final step of the algorithm. Remember from the previous section that $E(1)$ is generated from E_{NN}: $E(1)[i,j] = 1$ if $E_{NN}[i,j] = 1$; otherwise, $E(1)[i,j] = 0$. The popularity index associated with each instance is then calculated by summing the values in every column. (i.e. the sum of the 1^{st} column is the popularity index of the 1^{st} instance, the sum of the 2^{nd} column is the popularity index of the 2^{nd} instance and so forth.) The $E(1)$ matrix and the popularity indicies of our example is given in Figure 11.10. Note that $E(k)$ matrices are not necessarily symmetric, see that $E(1)$ of Figure 11.10 is not symmetric.

In our dataset example, Figure 11.10 produces the labeling order of the instances: $\{P_1$, P_2, $P_3\}$. In other words, in the first round we will ask our expert to label P_1 and place that label in the active pool. In that round, since the active pool contains only 1 *labeled-instance* it will be the closest labeled neighbor of every test instance and the estimates for all the test instances will be the same (the label of P_1).

In the second round, P_2 will be labeled by the expert and placed into a pool of "active" (i.e.

$$\begin{array}{rr|ccc}
 & & P_1 & P_2 & P_3 \\ \hline
 & P_1 & 0 & 1 & 0 \\
 & P_2 & 1 & 0 & 0 \\
+ & P_3 & 1 & 0 & 0 \\ \hline
 & Popularity: & 2 & 1 & 0
\end{array}$$

Figure 11.10: Popularity is the sum of the $E(1)$'s columns.

labeled) examples. This time the test instances will have 2 alternatives to choose their closest-neighbor from, hence the estimates will be either the label of P_1 or the label of P_2. Finally the expert will label P_3 and place it into the active pool. The change of the active pool is shown in Figure 11.11. Note that we only move from $Round_i$ to $Round_{i+1}$ if the stopping rules (described above) do not fire.

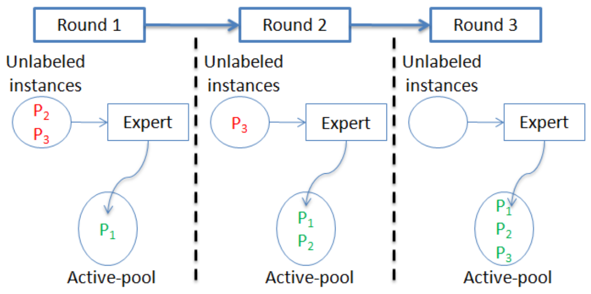


Figure 11.11: The change of active pool for the toy example. Note that in an actual setting transition between $Round_i$ to $Round_{i+1}$ is governed by the stopping rules.

11.5.6 Experimental Design

Our experimental rig has three parts:

1. Generate *baseline* results: Apply CART and passiveNN (passiveNN is the standard ABE0 algorithm) on the entire training set.

2. Generate the *active learning* results: Run QUICK on the active pool.

3. Compare the baseline results against results of QUICK.

1. Generate baseline results by applying CART and passiveNN on the entire training set. The algorithms are run on the entire training set and their estimations are stored. We use 10-way cross-validation:

- Randomize the order of instances in the dataset
- Divide dataset into 10 bins
- Choose 1 bin at a time as the test set and use the remaining bins as the training set
- Repeat the previous step using each one of the 10 bins in turn as the test set

2. Generate the active learning results by running QUICK. At each iteration firstly the features are selected and the active pool is populated with training instances in the order of their popularity. Training instances outside the active pool are considered unlabeled and QUICK is only allowed to use instances in the pool. Train and test sets are generated by 10-way-cross-validation.

Before a training instance is placed in the active pool, an expert labels that instance, i.e. the costing data is collected. When the active pool only contains 1 instance, the estimates will all be the same. As the active pool is populated, QUICK has more labeled training instances to estimate from.

3. Compare baseline to active learning: Once the execution of the algorithms is complete, the performance of QUICK, *passiveNN* and CART are compared under different performance measures. The QUICK estimates used for comparison are the ones generated by the active pool at the stopping point. An important point to note here is that QUICK has no relation with the derivation of the baseline results for *passiveNN* and CART. It is true that QUICK makes use of *k*-NN methods in its execution; however the derivation of baseline *passiveNN* and CART are separate procedures from the derivation of QUICK's. In the comparison phase, the results of these separate procedures are compared. Another point to note is that QUICK only uses the selected features (from the first execution phase) to decide which instances are to be labeled and placed into the active pool.

11.6 Results

11.6.1 Estimation Performance

Figure 11.12 is a sample plot for a representative data set (shown is the **desharnais** data set). It is the result of QUICK's synonym pruning (4 features selected for desharnais) followed by labeling N' instances in decreasing popularity. Following is the reading of Figure 11.12:

- Y-axis is the logged MdMRE error measures: the smaller the value the better the performance.
- The line with star-dots shows the error seen when i^{th} instance was estimated using the labels $1..(i-1)$.
- The horizontal lines show the errors seen when estimates were generated using all the data (either from CART or *passiveNN*).
- The vertical line shows the point where QUICK advised that labeling can stop (i.e. N').
- The square-dotted line shows *randNN*, which is the result of picking any random instance from the training set and using its effort value as the estimate.

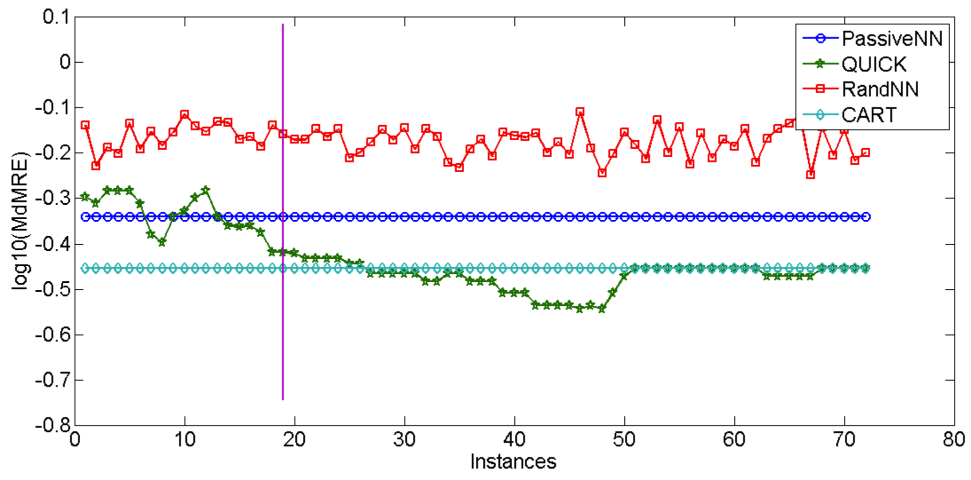


Figure 11.12: Sample plot of a representative (in that case desharnais) dataset showing the stopping point (the line parallel to the y-axis at $x = 19$) and MdMRE values (logged with base 10 for easier visualization). Note that QUICK uses only the selected 4 features of desharnis data set, whereas other methods use all of the 10 features.

Three observations of importance from Figure 11.12 are the (a) *reduction in the number of instances required*, (b) *the reduction in the number of features* and (b) *the estimation error*. With a fraction of the instances and the features of the original data set, QUICK is able to get as low error rates as CART and passiveNN, which both use entire data set. Furthermore, we have seen that the observed effect of QUICK is not random, as the *randNN* has a much higher prediction error. Possible other performance scenarios are: QUICK is statistically significantly (1) the same as CART and passiveNN, (2) better than passiveNN but worse than CART, (3) worse than passiveNN but better than CART and (4) worse than passiveNN and CART. To observe the possible scenarios we cannot make use of Figure 11.12 any more due to two reasons: 1) Repeating Figure 11.12 for *7 error measures* $\times$ *17 data sets* would be consuming too much space without conveying any new information; 2) more importantly, Figure 11.12 does not tell whether or not differences are significant; e.g. see Table 11.5 and Table 11.6 that performance difference of QUICK to passiveNN and CART are not significant for desharnais data set. Therefore, we provide summary analysis in the following subsections.

Table 11.4: The number of selected features (F') and selected instances (N') of a data set D with N instances and F independent features. QUICK uses $\frac{N'}{N}$ percent of the instances and $\frac{F'}{F}$ percent of the features for estimation. Thinking that each $N \times F$ intersection is a cell of D, the last column shows what percent of the original cells are used by QUICK. This table is sorted by the last column.

Dataset	N	N'	$\frac{N'}{N}$	F	F'	$\frac{F'}{F}$	$\frac{N'*F'}{N*F}$
nasa93	93	21	23 %	16	3	19 %	4 %
cocomo81	63	11	17 %	16	6	38 %	7 %
cocomo81o	24	13	54 %	16	2	13 %	7 %
maxwell	62	10	16 %	26	13	50 %	8 %
kemerer	15	4	27 %	6	2	33 %	9 %
desharnais	81	19	23 %	10	4	40 %	9 %
miyazaki94	48	17	35 %	7	2	29 %	10 %
desharnaisL1	46	12	26 %	10	4	40 %	10 %
nasa93_center_5	39	11	28 %	16	6	38 %	11 %
nasa93_center_2	37	16	43 %	16	4	25 %	11 %
cocomo81e	28	9	32 %	16	7	44 %	14 %
desharnaisL2	25	6	24 %	10	6	60 %	14 %
sdr	24	10	42 %	21	8	38 %	16 %
desharnaisL3	10	6	60 %	10	3	30 %	18 %
nasa93_center_1	12	4	33 %	16	9	56 %	19 %
cocomo81s	11	7	64 %	16	5	31 %	20 %
finnish	38	17	45 %	7	4	57 %	26 %
albrecht	24	9	38 %	6	5	83 %	31 %

11.6.2 Reduction in Sample and Feature Size

Table 11.4 shows reduction results from all 18 data sets used in this study. The N column shows the size of the data and the N' column shows how much of that data was labeled by QUICK. The $\frac{N'}{N}$ column expresses the percentage ratio of these two numbers. In a similar fashion, F shows the number of *independent features* for each data set, whereas F' shows the number of selected features by QUICK. The $\frac{F'}{F}$ ratio expresses the percentage of selected features to the total number of the features. The important observation of Table 11.4 is that, given N projects and F features, it is neither necessary to collect detailed costing details on 100% of N nor is it necessary to use all the features. For nearly half the data sets studied here, labeling around one-third of N would suffice (median of $\frac{N'}{N}$ for 18 data sets is 32.5%). There is a similar scenario for the amount of features required. QUICK selects around one-third of F for half the data sets. The median value of $\frac{F'}{F}$ for 18 data sets is 38%.

The combined effect of synonym and outlier pruning becomes more clear when we look at the last column of Table 11.4. Assuming that a data set D of N instances and F independent features is defined as an N-by-F matrix, the reduced data set D' is a matrix of size N'-by-F'. The last column shows the total reduction provided by QUICK in the form of a ratio: $\frac{N'*F'}{N*F}$. The rows of Table 11.4 are sorted according to this ratio. Note that the maximum size requirement (albrecht data set) is only 32% of the original data set and with QUICK we can go as low as only 4% of the actual data set size (nasa93). This is an important observation regarding the intrinsic complexity of SEE data sets. From the above result we are able to conclude that the reduced data sets (D') proposed for discussion adequately represent all the project cases available in the dataset, given the resultant differences are statistically insignificant.

11.6.3 Comparison QUICK vs. CART

Figure 11.13 shows the PRED(25) difference between CART (using all the data) and QUICK (using just a subset of the data). The difference is calculated as *PRED(25) of CART minus PRED(25) of QUICK*. Hence, a *negative* value indicates that QUICK offers better PRED(25) estimates than CART. The left hand side (starting from the value of -35) shows QUICK is better than CART, whereas in other cases CART outperformed QUICK (see the right hand side until the value of +35).

From Figure 11.13 we see that 50^{th} percentile corresponds around PRED(25) value of 2, which means that at the median point the performance of CART and QUICK are very close. Also note that 75^{th} percentile corresponds to less than 15, meaning that for the cases when CART is better than QUICK the difference is not dramatic. Our results show that the value-added in using all the projects and features is limited. A QUICK analysis of just a small percentage of the data can yield estimates as good as the more complex learners like CART.

11.6.4 Detailed Statistical Analysis

Table 11.5 and Table 11.6 compare QUICK to *passiveNN* and CART using seven evaluation criteria. *Smaller* values are *better* in these tables. When calculating "loss" for six of the measures, "loss" means higher error values. On the other hand, for PRED(25), "loss" means lower values. The last column of each table sums the losses associated with the method in the related row.

By sorting the values in the last column of Table 11.5, we can show that the number of losses is very similar to the nearest neighbor results:

$$\begin{aligned} QUICK: &\quad 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,6 \\ passiveNN: &\quad 0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,3,6,6,7 \end{aligned}$$

The gray rows of Table 11.5 show the datasets where QUICK loses over most of the error measures (4 times out of 7 error measures, or more). The key observation here is that, when using nearest neighbor methods, a QUICK analysis loses infrequently (only 1 gray row) compared to a full analysis of all projects.

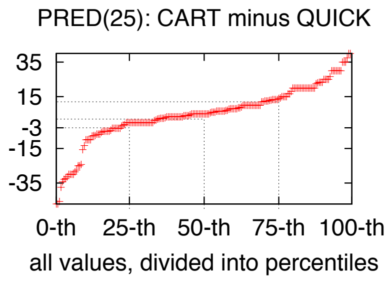


Figure 11.13: CART minus QUICK values for PRED(25) in 18 data sets. Negative values mean that QUICK outperformed CART. The median is only 2% with a small interquartile range ($75^{th} - 25^{th}$ percentile) of 15%.

As noted in Figure 11.13, QUICK has a close performance to CART. This can also be seen in the number of losses summed in the last column of Table 11.6. The 4 gray rows of Table 11.6 show the data sets where QUICK loses most of the time (4 or more out of 7) to CART. In just $4/18 = 22\%$ of the data sets is a full CART analysis better than a QUICK partial analysis of a small subset of the data. The sorted last column of Table 11.6:

$$\begin{aligned} QUICK: &\quad 0,0,0,0,0,0,0,0,0,0,0,0,0,1,3,6,7,7,7 \\ CART: &\quad 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 \end{aligned}$$

11.7 Discussion

An important point of discussion is how an industry practitioner can use the proposed approach. An example application process would be as follows:

1. Identify dependent and independent variables.
2. Decide the importance and easiness-to-collect of the independent variables.
3. Start collecting important and easy-to-collect independent variables.
4. Run QUICK to decide which of these features to keep.
5. Run QUICK to find popular projects, i.e. which would be used by ABE0-1NN.
6. Collect dependent variables of only the projects marked in the previous step.

One last remark regarding QUICK is its contradictory results with pre-experimental expectations. The number of instances contained within a data set is fundamental to the information content of this data set. Hence, the expectation is that unless the instances are identified as statistical outliers, their removal will destabilize a prediction model and decrease the estimation performance. On the other hand, QUICK is demonstrated to work as good as passiveNN and CART. This can be explained by the underlying simplicity of the data. Except for a core set of instances, the rest can safely be removed.

Table 11.5: QUICK (on the reduced data set) vs. passiveNN (on the whole data set) w.r.t. number of losses, i.e. *lower* values are *better*. Right-hand-side column sums the number of losses. Rows highlighted in gray show data sets where QUICK performs worse than passiveNN in the majority case (4 times out of 7, or more). Note that only 1 row is highlighted.

Dataset	**Method**	**MMRE**	**MAR**	**Pred(25)**	**MdMRE**	**MBRE**	**MIBRE**	**MMER**	**SUM OF LOSSES**
cocomo81	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
cocomo81e	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
cocomo81o	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
cocomo81s	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnais	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL1	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL2	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL3	passiveNN	0	0	0	0	0	0	0	0
	QUICK	1	0	1	1	1	1	1	6
nasa93	passiveNN	0	0	0	0	1	1	0	2
	QUICK	0	0	0	0	0	0	0	0
nasa93_center_1	passiveNN	1	0	1	1	1	1	1	6
	QUICK	0	0	0	0	0	0	0	0
nasa93_center_2	passiveNN	1	1	1	1	1	1	1	7
	QUICK	0	0	0	0	0	0	0	0
nasa93_center_5	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
sdr	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
albrecht	passiveNN	1	1	1	1	1	1	0	6
	QUICK	0	0	0	0	0	0	0	0
finnish	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
kemerer	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
maxwell	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	1	1	1	3
miyazaki94	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0

11.8 Conclusion

The goal of this research was to investigate the essential content of SEE data sets and make recommendations regarding which estimation methods (simple or complex) should be favored. We defined the essential content as the number of $F' \subseteq F$ features and $N' \subseteq N$ instances required to hold the information of a data set.

Table 11.6: QUICK (on the reduced data set) vs. CART (on the whole data set) w.r.t. number of losses, i.e. *lower* values are *better*. Right-hand-side column sums the number of losses. Gray rows are the data sets where QUICK performs worse than CART in the majority case (4 times out of 7, or more).

Dataset	Method	MMRE	MAR	Pred(25)	MdMRE	MBRE	MIBRE	MMER	SUM OF LOSSES
cocomo81	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
cocomo81e	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
cocomo81o	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
cocomo81s	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnais	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL1	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL2	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
desharnaisL3	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	1	1	1	3
nasa93	CART	0	0	0	0	0	0	0	0
	QUICK	0	1	0	0	0	0	0	1
nasa93_center_1	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
nasa93_center_2	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
nasa93_center_5	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
sdr	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
albrecht	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
finnish	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
kemerer	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
maxwell	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
miyazaki94	CART	0	0	0	0	0	0	0	0
	QUICK	1	0	1	1	1	1	1	6

Our results showed that essential content of SEE data sets is surprisingly small. Even the most commonly studied data sets (e.g. **nasa93** data set which could be summarized only by 4% of its actual size) could be summarized by a small portion of their features and instances. We also saw that such a reduction protects the estimation performance. The implications of our research is two-folds:

1. SEE data sets can be reduced to a small essential content and fortunately simple methods are

Table 11.7: QUICK's sanity check on 8 company data sets (company codes are C1, C2, ..., C8) from Tukutuku. Cases where QUICK is loses for majority of the error measures (4 or more) are highlighted. QUICK is statistically significantly the same as CART for 5 out of 8 company data sets for majority of the error measures (4 or more). Similarly, QUICK is significantly the same as passiveNN for 5 out of 8 company data sets.

Dataset	Method	MMRE	MAR	Pred(25)	MdMRE	MBRE	MIBRE	MMER	SUM OF LOSSES
C1	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
C2	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
C3	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C4	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C5	CART	1	0	1	1	0	0	0	3
	QUICK	0	0	0	0	0	0	0	0
C6	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C7	CART	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
C8	CART	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C1	passiveNN	0	0	1	0	0	0	0	1
	QUICK	1	1	0	1	1	1	0	5
C2	passiveNN	0	0	1	0	0	0	0	1
	QUICK	1	1	0	1	1	1	0	5
C3	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C4	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C5	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C6	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0
C7	passiveNN	0	0	0	0	0	0	0	0
	QUICK	1	1	1	1	1	1	1	7
C8	passiveNN	0	0	0	0	0	0	0	0
	QUICK	0	0	0	0	0	0	0	0

still able to perform well on the essential content.

2. QUICK can help to identify the important features and instances.

Therefore, at the end of the research presented in this chapter, we recommend the following principle:

Principle #11: Combine Outlier and Synonym Pruning

The contributions of this research can be summarized as follows:

- An unsupervised method to find the essential content of SEE data sets and reduce the data needs.
- Promoting research to elaborate on the data, not on the algorithm.

Chapter 12

Principle #12, Be Aware of Sampling Method Trade-off

This chapter targets a more theoretical aspect of SEE research compared to the prior chapters. In this chapter we will address the choice of sampling methods (SM) used in various SEE studies. In that regard, this chapter targets a more research oriented audience than a practical audience. However, the implications of the research presented in this chapter is equally important both for practitioner as well as researchers. The reason being is that the case for a new estimation method is usually made by comparing the new method to existing ones through evaluation of existing data sets. These data sets are separated into training and test sets through the use of SMs. The choice of the appropriate SM is a matter of trade-off between associated bias and variance values, run times and the ease of replication. In order to evaluate the trade-off between these factors, it is essential to have an extensive empirical experimentation, hence the rest of this chapter. The proper evaluation of the trade-off and a consensus on the right SM to use in SEE studies is also likely to decrease the fact that different research groups come with different conclusions. The implication of a consensus on the right SM to use is likely to reflect on the recommendations to practitioner audiences.

Assessing new prediction methods is complicated by small sample sizes of the training data. Valerdi [174] and Hihn [26] offer the rule of thumb that there should be five to ten rows of training data per attribute. Most effort estimation data sets are smaller than that: for example, five recent effort estimation publications [1, 16, 46, 61, 70] use data sets with dozens of attributes but only a handful of rows (median values of the number of rows are 13, 15, 31, 33, 52, respectively).

As a result, prediction models tend to be overfitted to the particulars of the training data used in particular studies. This leads to the problem of *conclusion instability*; i.e. different studies make different conclusions regarding what is the “best” effort estimator. Shepperd et al. [4, 14] studied a large number of synthetic data sets (generated from distributions found in a real-world data set). They found that, as they changed the conditions of their experiments, no method was consistently best across every condition. Specifically, the performance of a method depends on:

1. The dataset;
2. The evaluation method used to assess model accuracy;
3. The generation method used to build training and test sets.

Elsewhere, we have addressed points #1 and #2 [49, 148][1]. This chapter focuses on the third point listed above; i.e. the *generation method used to build training and test sets*. SEE research uses historical data to estimate future performance. The induced predictor is tested on data that is not used in generating the predictor. In practice, this means using some sampling method (SM) to divide historical data into:

- Training data that the prediction system can learn from
- Unseen or test validation data that is used to assess predictive accuracy.

SEE validation studies adopt different SMs such as leave-one-out (LOO), 3Way and 10Way [45, 76, 175, 176]. The difference between these SMs is as follows:

LOO:

Take one instance at a time as the test set

Build the learner on the remaining $N - 1$ instances (training set)

Use the model to estimate for the test set.

[1] If R_i is the rank of method M_j within a set of M methods, then we use δr to denote the maximum rank change of that method as we alter the evaluation method and the data set. Our own experiments confirmed Shepperd's previous work; i.e. that $\delta r \neq 0$. Hence, we cannot say for certain that a method ranked at R_i is always better than another ranked at R_{i+1}. However, we have found that if we analyzed enough methods using enough evaluation methods, then the ranking variability is much smaller than the number of methods; i.e. for large M, $\delta r \ll |M|$. That is, given 90 methods, we can say that (a) the top 30 are better than the bottom 30; (b) the value of the methods ranked 31 to 59 is unknown; so (c) we should focus on those methods in the top third [49]. Other experiments have confirmed the superiority of those top ranked methods [148].

N-Way:

```
Randomize order of rows in data
Divide dataset into $N$ subsets of size close or equal to $1/N$
Use each subset as the test set and the remaining subsets as the training set
Repeat this procedure multiple times: Hall & Holmes recommend ten repeats of a 10-way study [56].
```

As shown in Figure 12.1, there is no consensus in the effort estimation literature as which SM should be used to evaluate new predictors. Note that, in that figure, many researchers use some variant of N-way. We argue that the use of N-way contributes significantly to the conclusion instability problem. The randomization step of N-way makes the results virtually unrepeatable since a sequence of random numbers are usually very different when generated by different algorithms implemented in different languages running different toolkits on different platforms. This means that an N-way analysis incurs the problem of point #3, discussed above.

Method	**Used by**
LOO	[13, 16, 19, 24, 38, 74, 105, 157]
Others (ad-hoc, 6-Way etc.)	[26, 28, 46, 90, 111, 158]
10-Way	[16, 32, 122, 162]
3-Way	[16, 177, 178]

Figure 12.1: SEE papers that use different SEE methods.

On the other hand, a LOO analysis is deterministic and repeatable since, given access to the same data, it is possible to generate identical train and test sets. However, there are two problems with LOO: the *high variance* of LOO and its *long runtimes*:

- *High variance:* In theory, as discussed below, the results of a LOO analysis will have a different (and higher) variance and bias than a N-way study. This introduces a complication into the analysis; e.g. using LOO it will be harder to distinguish the performance of different methods since the performance of those methods will exhibit a wider variability.

- *Long runtimes:* A 3-way analysis of 1000 examples will require the construction of three effort models. On the other hand, a LOO analysis of the sample examples will require the construction of 1000 effort models. If the effort model is slow to generate (e.g. some genetic algorithm exploring all subsets of possible attributes [46]) then 1000 repeats is impractically slow.

We show in this chapter that neither of these problems are necessarily an issue:

- For 20 data sets from the PROMISE repository, we show that the variance (and the bias) of results generated by LOO is statistically indistinguishable from 10-way and 3-way. That is, while in theory LOO and N-way studies generate different results, in practice, those differences are insignificant.
- LOO conducts many repeated calculations over the same data sets. If those calculations are cached and re-used later in the LOO, then the runtimes for LOO become close to those of N-way.

The conclusion of this chapter will be that there is no reason from the effort estimation community to suffer with N-way studies:

- The deterministic nature of LOO studies makes them more repeatable.
- They are not necessarily slower than N-way.
- Nor do they generate different biases and variances.

Therefore, for SEE research we recommend LOO, at the very least for the SEE research using the data sets studied in this chapter. A more general recommendation we have is the principle endorsed in this chapter, i.e. to *"be aware of the bias-variance, run-time and replication trade-off"* inherent in different SMs.

12.1 Defining Bias & Variance

A typical SEE dataset consists of a matrix X and a vector Y. The input variables (a.k.a. features) are stored in X, where each row corresponds to an observation and each column corresponds to a particular variable. Similarly, the dependent variable is stored in a vector Y, where for each observation in X there exists a response value.

Now assume that a prediction model represented by $\hat{f}(x)$ has been learned from a training dataset and the actual values in the training set were generated by an unknown function $f(x)$. So as to measure the errors between the actual values in Y and the predictions given by $\hat{f}(x)$, we can make use of an error function represented by $L(Y, \hat{f}(x))$. Some examples of error functions are squared loss (Equation 12.1) and absolute loss (Equation 12.2).

$$L(Y, \hat{f}(x)) = \left(Y - \hat{f}(x)\right)^2 \tag{12.1}$$

$$L(Y, \hat{f}(x)) = |Y - \hat{f}(x)| \tag{12.2}$$

Given the assumptions that the underlying model is $Y = f(X) + \epsilon$ and $Var(\epsilon) = \sigma_\epsilon^2$, we can come up with a derivation of the squared-error loss for $\hat{f}(X)$ [77], where $f(X)$ is the underlying true model and $\hat{f}(X)$ is the learned model through an SM. The error for a point $X = x_0$ is:

$$\begin{aligned}
Error(x_0) &= E\left[\left(Y - \hat{f}(x_0)\right)^2 |X = x_0\right] \\
&= \sigma_\epsilon^2 + \left(E[\hat{f}(x_0) - f(x_0)]\right)^2 + E\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right] \\
&= \sigma_\epsilon^2 + Bias^2(\hat{f}(x_0)) + Var(\hat{f}(x_0)) \\
&= \underbrace{IrreducableError}_{1^{st}Term} + \underbrace{Bias^2}_{2^{nd}Term} + \underbrace{Variance}_{3^{rd}Term}
\end{aligned}$$

In the above derivation, the explanations of the 1^{st}, 2^{nd} and 3^{rd} terms are as follows:

- The $1^{st}Term$ is the so called *"irreducible error"*, i.e. the variance of the actual model around its true mean. This variance is inevitable regardless of how well we model $f(x_0)$, only exception to that is when the actual variance is zero (when $\sigma_\epsilon^2 = 0$).
- The $2^{nd}Term$ is the square of the bias, which is the measure of how different the model estimates are from the *true* mean of the underlying model.
- The $3^{rd}Term$ is the variance of the estimated model. It is the expectation of the squared deviation of the estimated model from its own mean.

Seni and Elder warns about the calculation of bias that it *cannot* be computed but can be used as a helpful theoretical concept (see p. 23 of [76]). The biggest handicap towards calculation of bias comes from the fact that we can never know the true model [76,179], unless it was designed as a mathematical model in the first place. Then we cannot derive the bias (see the true model, $f(x_0)$, in the derivation). That is a critical problem that needs to be handled, if we are to make empirical investigations on the $B\&V$ trade-off in SEE. Without concrete definitions of "bias"; hence the "true model", $B\&V$ discussions regarding sampling methods in SEE will be nothing more than expert opinions.

To handle that problem, we need to make assumptions regarding the true model. A successful application of such an assumption is provided by Molinaro et al.: A learner trained on the whole dataset is taken as the true model [179]. This approach is quite useful as it replaces the *unknown* true model with a *known*, mathematically definable model; thereby, enabling the bias derivation. In our experimentation we used the option provided by Molinaro et al. [179].

12.2 In Theory, SMs Affect Results

An important question associated with SMs is how $B\&V$ relate to different choices of the training size (*K*). To answer this question, we make two observations. Given a data set D of fixed size, and test and training data sets $D = train \cup test$, then:

- The training set grows progressively smaller from LOO to 10way to 3way.
- The test set grows progressively larger from LOO to 10way to 3way.

The first observation effects the bias and the second observation effects the variance. To see that, recall that any induction algorithms seeks for a target concept in some training data. As a training set gets smaller, it becomes less likely to contain examples that describe the target. Hence, the induction algorithm will "miss" the target and the resulting model will be biased (its predictions will deviate away from true predictions). That is, in theory, bias will increase from LOO to 10way to 3way. This theoretical prediction is endorsed by Kitchenham and Mendes who have discussed the effects of SMs on $B\&V$ in the context of cross-within-company data [21]. They write that LOO biases positively towards within-company data (but they do not confirm that with experimentation).

On the other hand, as the training set shrinks, the test set grows. To understand the effect of test set size N on the variance, recall that variance is the difference between each prediction and the mean prediction; i.e.

$$Var(\hat{f}(x_0)) = E\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right] = \frac{\sum_i^N (X_i - \mu)}{N}$$

Note that $\lim_{N\to 0} Var(\hat{f}(x_0)) = \infty$; i.e. smaller tests sets can have larger variance. Hence, in theory, variance will increase from 3way to 10way to LOO.

In summary, according to the above discussion, we would expect:

- LOO : High variance, low bias (see upper left of Figure 12.2)
- 3Way : Low variance, high bias (see lower right of Figure 12.2)
- 10Way : Values between LOO and 3Way (see center of Figure 12.2)

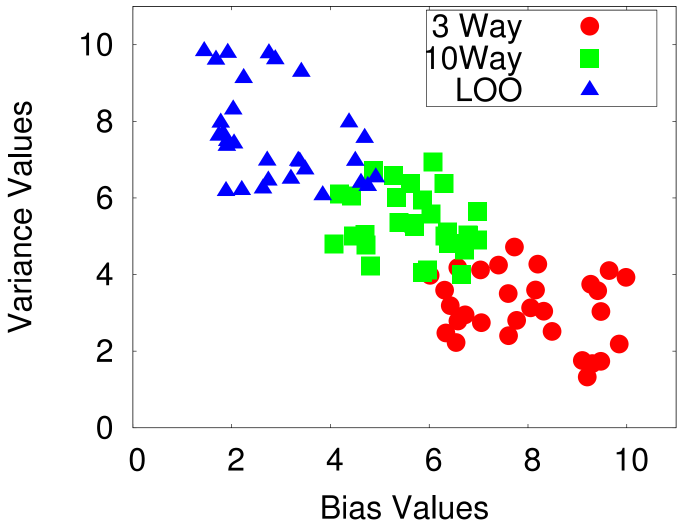


Figure 12.2: A simple simulation for the *"expected"* case of $B\&V$ relation to testing strategies.

The results of this chapter can be expressed with respect to Figure 12.2: the empirical results (reported below) cannot distinguish between the $B\&V$ of LOO, 3Way, and 10Way.

12.3 Experiment1: Comparing Bias & Variance

12.3.1 Experiments

Generate True Model $f(x)$: Each algorithm is trained on each entire dataset and the estimates are stored as the values of $f(x)$. The values of $f(x)$ will be used for $B\&V$ calculations.

Get estimates: Let A_i ($i \in \{1, 2, ..., 90\}$) be one of the 90 algorithms (as introduced in §2.4) and let D_j ($j \in \{1, 2, ..., 20\}$) be one of the 20 public datasets (of Figure 2.1 in §10.1). Also let SM_k ($k \in \{1, 2, 3\}$) be one of the 3 SMs. In this step every A_i is run on every D_j subject to every SM_k. In other words every $A_i \times D_j \times SM_k$ combination is exhausted, and related predictions are stored to be used for $B\&V$ calculations.

Calculate $B\&V$ values: The $f(x)$ values and predictions coming from $A_i \times D_j \times SM_k$ runs are used to calculate the $B\&V$. At the end of this step, we have separate $B\&V$ values for every $A_i \times D_j \times SM_k$. Another interpretation is that for every algorithm-dataset combination ($A_i \times D_j$) we have 3 values of $B\&V$ (1 for each SM_i).

Statistical Check on $B\&V$ values: In this step we check if the $B\&V$ values for every $A_i \times D_j$ combination are statistically different from one another (checks are based on Mann-Whitney at 95% confidence interval). This way we can see if the run of an algorithm on a single dataset subject to different SMs generate significantly different $B\&V$ values. Since we have 3 different

```
for $D_j$, where $j \in 1...20$ do
  for $A_i$, where $i \in 1...90$ do
    for $Tupple$, where $Tupple \in$ {LOOvs3Way; LOOvs10Way; 3Wayvs10Way} do
      if Mann-Whitney(Bias values of $A_i$ for $D_j$ from SM's in $Tupple$, 95%) says the same then
        Mark method as "tied" for bias values
      else
        Mark method as "not-tied" for bias values
      end if
      if Mann-Whitney(Var. values of $A_i$ for $D_j$ from SM's in $Tupple$, 95%) says the same then
        Mark method as "tied" for variance values
      else
        Mark method as "not-tied" for variance values
      end if
    end for
  end for
end for
```

Figure 12.3: Comparing $B\&V$ values coming from different $A_i \times D_j$ combinations under different SM tuples. This comparison helps us see what percentage of 90 methods "tie" w.r.t. $B\&V$ values.

SMs, for every $A_i \times D_j$ there are 3 different tuples to look at: LOO vs. 3Way; LOO vs. 10Way; 3Way vs. 10Way. For each tuple we ask Mann-Whitney if the $B\&V$ values coming from $A_i \times D_j$ are different under the SM's of that tuple. We note down whether they are statistically the same (i.e. they "tie") or not. After processing all the SM tuples for all $A_i \times D_j$, we can see what percent of the 90 algorithms generated statistically same $B\&V$ values for different SM tuples and for different datasets. The pseudo-code for this process is given in Figure 12.3.

12.3.2 Results

After calculating the $B\&V$ values for 90 algorithms on all the datasets, we were unable to observe the behavior of Figure 12.2, i.e. we did not observe three clusters at predicted $B\&V$ zones. On the contrary, we observed that $B\&V$ values associated with different SMs were very close to one another.

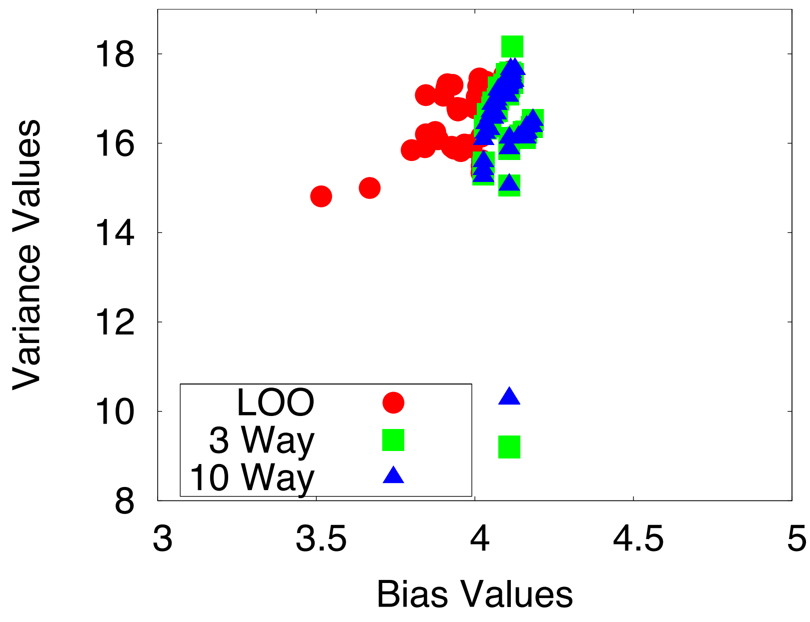


Figure 12.4: $B\&V$ values for china data set (shown values are the natural logarithm of the actual values).

For example, see in Figure 12.4 the mean $B\&V$ values of 90 algorithms for china data set. Note that different SMs are represented with different symbols and for every SM there are 90 symbol occurrences corresponding to 90 algorithms. The $B\&V$ values associated with each SM overlap, instead of forming separate clusters. Also, the *expected* relative low and high $B\&V$ values of SMs (see Figure 12.2 for expected low and high) were not visible too. Unlike the expected behavior, the actual $B\&V$ values were both high, regardless of the utilized SM.

We have conducted these experiments on all the pubic datasets and generated Figure 12.4 for every dataset. However, the results are the same:

1. The *expected* behavior was not found LOO, 3-Way and 10Way;

2. The different SMs did not form distinct clusters (as witnessed by the overlapping $B\&V$ values of LOO, 3-Way and 10Way in Figure 12.4).

There is insufficient space to repeat Figure 12.4 for every dataset. Hence, we summarized these $B\&V$ values in terms of quartile charts of Figure 12.5. Figure 12.5 shows every dataset in a separate row, which is then divided into 3 sub-rows. Sub-rows correspond to 3 different SMs and they show the related $B\&V$ quartile charts separately. In every quartile chart, the median (represented with a dot), 25^{th} quartile (the left horizontal line-end) and the 75^{th} quartile (the right horizontal line-end) are shown. Note that all of Figure 12.4 appears as the last 3 rows of Figure 12.5.

Figure 12.6 shows what percent of 90 algorithms "tied" w.r.t. to Mann-Whitney; i.e. difference in their $B\&V$ values were statistically indifferent. See Figure 12.3 for the pseudo-code of the comparison of $B\&V$ values w.r.t. Mann-Whitney. Every cell of Figure 12.6 reports the percentage of methods (out of 90) that "tied" for particular SM tuples (LOO vs. 3Way, LOO vs. 10Way, 3Way vs. 10Way) under different datasets.

The distributions of Figure 12.6 are summarized in Figure 12.7: note the high number of ties. That is, measured in the number of ties as witnessed by Mann-Whitney, these results were the same more often than not.

In order to better explore the deltas between our treatments, we applied a 1-way ANOVA analysis. 1-way ANOVA test takes a vector of output values (bias and variance one at a time) and a factor, which in our case is the sampling method. The p-value yielded by this 1-way ANOVA tests the null hypothesis that all samples are drawn from populations with the same mean. If a p-value is near zero, then this casts doubt on the null hypothesis, i.e. at least one of the sample means is significantly different than the others. The p-value for bias values is: 0.107, which is a

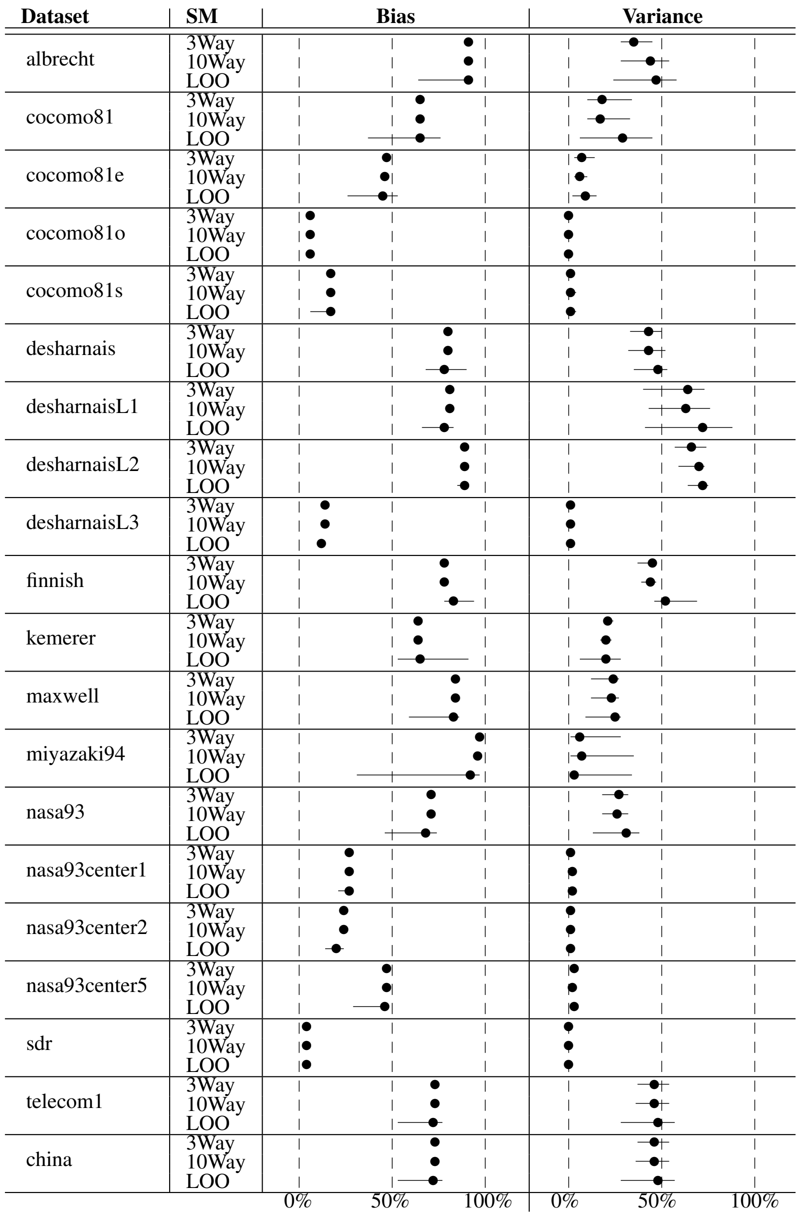


Figure 12.5: $B\&V$ values in quartiles for all datasets. Black dots denote median values. Horizontal lines denote the inter-quartile range (25 to 75 percentile band). In many results, the inter-quartile range is so small that it disappears behind the median dots. For the purposes of display, all the bias values were normalized min to max (of each dataset row), 0 to 100 (ditto with variance). The key observation from this result is that within each group of SMs, the $B\&V$ are very similar.

dataset		bias		variance	
		3Way	10Way	3Way	10Way
cocomo81	LOO	53	61	61	61
	3Way	-	90	-	61
cocomo81o	LOO	74	81	70	73
	3Way	-	91	-	50
cocomo81e	LOO	62	61	48	50
	3Way	-	86	-	53
cocomo81s	LOO	59	58	48	48
	3Way	-	40	-	28
nasa93	LOO	47	46	60	64
	3Way	-	90	-	71
nasa93_center_1	LOO	72	74	40	43
	3Way	-	92	-	50
nasa93_center_2	LOO	56	54	49	58
	3Way	-	72	-	48
nasa93_center_5	LOO	58	61	66	61
	3Way	-	94	-	69
desharnais	LOO	81	81	80	78
	3Way	-	100	-	92
desharnaisL1	LOO	79	79	79	79
	3Way	-	100	-	83
desharnaisL2	LOO	89	90	77	82
	3Way	-	99	-	79
desharnaisL3	LOO	79	80	60	71
	3Way	-	94	-	47
sdr	LOO	39	40	39	37
	3Way	-	2	-	22
albrecht	LOO	80	82	73	80
	3Way	-	80	-	69
finnish	LOO	83	83	83	83
	3Way	-	100	-	83
kemerer	LOO	70	64	56	53
	3Way	-	100	-	77
maxwell	LOO	67	73	87	88
	3Way	-	92	-	70
miyazaki94	LOO	38	46	30	38
	3Way	-	64	-	48
telecom	LOO	84	82	78	72
	3Way	-	100	-	76
china	LOO	40	46	53	52
	3Way	-	80	-	58

Figure 12.6: Percentage of algorithms for which $B\&V$ values coming from different SMs are the same (according to Mann-Whitney at 95). Note the very high percentage values, meaning that for the majority of the algorithms different SMs generate statistically the same values.

border value for a significance level of 99%. Similarly, p-value for the variance values is: 0.348 for a significance level of 99%.

Due to: 1) the border p-value of ANOVA and 2) the fact that ANOVA assumes sample distributions are Gaussian (which is hardly the case for SEE datasets[2]), we also performed a Friedman test (which is a rank-based non-parametric test) followed by a multiple comparison test. Friedman's test is appropriate when the assumptions of a parametric test do not hold and when we are interested in the effects of treatments (represented by columns) under study [180]. In our study columns represent the $B\&V$ values (separately) coming from 3 different SM's. The Friedman test compares the means of multiple groups to test the hypothesis that *"they are all the same"*,

[2]To see the non-Gaussian behavior of $B\&V$ values, refer to http://goo.gl/HH5b9 where $B\&V$ values are plotted in the form of 10-bin equal-width histograms for each SM and dataset.

		Percentile		
	Comparison	25th	50th (median)	75th
Bias	LOO vs 3Way	53	67	79
	LOO vs 10Way	58	69	81
	3way vs 10Way	80	91	99
Variance	LOO vs 3Way	48	60	77
	LOO vs10way	48	61	77
	3Way vs 10Way	50	64	76

Figure 12.7: Percentiles of number of ties from Figure 12.6.

versus they are not all the same. When this differentiation is too general, i.e. when we want to see further information regarding which pairs of means are different and which are not, we follow the Friedman test with a rank-based multiple comparison procedure [181]. The confidence level used for the multiple comparison test is 99%. Based on the selected confidence interval, multiple comparison test calculates the span of confidence interval in terms of ranks around the mean rank value of each SM. The expectation is that for two sample means to be statistically different, their span of confidence interval around the mean rank should form disjoint sets [3]. Figure 12.8 shows the results of the multiple comparison test. The x-axis of this figure shows the average ranks (according to bias and variance values, separately) corresponding to different sampling methods. The y-axis shows the ID's of the sampling methods (1 is for LOO, 2 is for 3Way and 3 is for 10Way). The mean ranks of each sampling method is represented with a symbol and an interval around the symbol. The interval -so called comparison interval- shows the span of the confidence interval for each SM (the selected confidence interval here is 99%). Two means are statistically different from one another if their comparison intervals are disjoint. As can be seen in Figure 12.8, none of the SM's has a disjoint comparison interval, i.e. none of the SM's is significantly different.

[3]For actual $B\&V$ values see the csv file at http://goo.gl/C6dkF. In this link you will also find a "readme" file explaining the contents. The code to generate Figure 12.8 is given in this link too. For further implementation details regarding $friedman$ and $multcompare$ functions of MATLAB, refer to [180, 181] as well as related Mathworks tutorials.

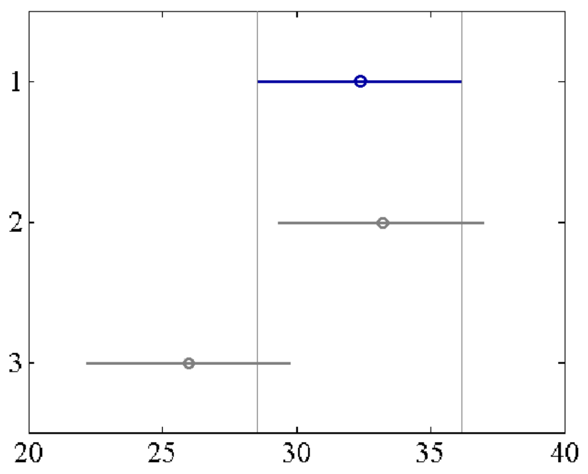


Figure 12.8.A : bias

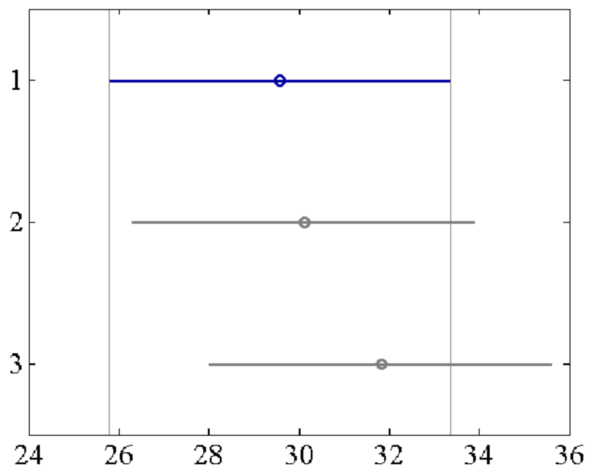


Figure 12.8.B : variance

Figure 12.8: Friedman test followed by multiple-comparison test. The x-axis shows the average ranks, whereas the y-axis shows the ID's of the SM's: 1 is for LOO, 2 is for 3Way and 3 is for 10Way. Two means are statistically different from one another if their comparison intervals are disjoint. None of the SM's has a disjoint comparison interval.

12.4 Experiment2: Reducing the Run-times for LOO

The total execution time of the experimentation is associated with a particular implementation method, i.e. different implementations of the same algorithm will have different run times. Therefore, we used standard MATLAB functions in this study: All methods except ABE0-1NN and ABE0-5NN, and all pre-processors except discretizers are found in MATLAB libraries.

The run times are also expected to be greatly affected by particular SMs. Each SM dictates a different number of times a learner is trained. The training-time of a learner is much greater than the testing-time since, once a learner is trained, the prediction for a particular test instance can be very quick. Below are the number of training times required for each SM on a *single* dataset:

- **LOO:** *N trains* where N is the dataset size.
- **3Way:** *10 repeats x 3 bins = 30 trains*
- **10Way :** *10 repeats x 10 bins = 100 trains*

For 20 datasets in this study (a total of 1198 instances), we expect the following training times:

- **LOO:** 1198 trains.

- **3Way:** 30 trains/dataset $\times 20$ datasets $= 600$ trains
- **10Way:** 100 trains/dataset $\times 20$ datasets $= 2000$ trains

From above number of training times, we expect 3Way to be the fastest SM, followed by LOO and 10Way.

In Figure 12.9 we see run times for 20 datasets. Our expectation that 3Way would be the fastest SM followed by LOO, then 10Way holds. However, the difference is much smaller than expected. Although there are orders of magnitude differences between SMs in terms of train-times, the run time difference is limited to a couple of minutes. Therefore, run time is not a critical factor in the choice of SMs.

SM	**Run time**
LOO	$9,960$
3Way	$9,360$
10Way	$10,140$

Figure 12.9: The run times in seconds. The expected order of SMs from fastest to lowest (3Way, LOO, 10Way) holds, however the difference is in the order of minutes.

One word of caution is that ABE0-xNN variants are the slowest methods in our experiments and if such slow methods are to be employed with LOO, then they need to be implemented carefully. Initially we coded ***ABE0-1NN*** and ***ABE0-5NN*** without regard to optimization: i.e. for every test instance, its distance to all the training instances were calculated from scratch. This way of brute-force implementation skyrockets the run times associated with LOO. The solution is to calculate the distance matrix of a dataset only once and cache that for future uses in LOO. The run time of the cached implementation for LOO -as given in Figure 12.9- is $9,960$ seconds, whereas the brute-force implementation is $25,920$ seconds. So the caching strategy decreases the run time of LOO by orders of magnitude.

12.5 Conclusions

To the best of our knowledge, in the field of SEE, this is the first empirical investigation on $B\&V$ and runtime trade-off inherent in different SMs.

Our experimentation investigates a large space of 90 algorithms and 20 datasets. The results present the surprising finding that $B\&V$ values in SEE domain behave quite different than the expected:

- Measured in terms of $B\&V$, different SMs are statistically the same.
- Similarity of SMs also persists in terms of the run times. See in Figure 12.9 that the biggest run time difference is between 3Way and 10Way, which is 780 seconds (13 minutes) or only a 7% difference in runtimes between the methods. However, note that some coding techniques(e.g. caching distance results in ***ABE0-xNN*** variants) can significantly lower LOO run times.

Thus, for SEE research (particularly for the experiments that target the public data sets studied in this chapter), we recommend the use of LOO. A more general recommendation could be summarized with the following principle:

$$\boxed{\text{Principle \#12: Be Aware of Sampling Method Trade-off}}$$

The contributions of the research presented in this chapter are:

- The first systematic investigation of *B&V* trade-off in SEE domain
- An extensive experimentation with 20 public datasets and 90 algorithms
- Showing that *B&V* trade-off and run times of SMs are not the main concerns for SEE
- Recommendation based on experimental concerns:
  - For reproducibility, we prefer LOO since this avoids non-deterministic selection of train and test sets.

Chapter 13

Final Remarks

Until now we shared our experiences regarding data collection in industrial settings as well as our rigorous experimentation for the open issues of the SEE field. We summarized our findings in the form of 12 principles and presented the related contributions at the end of each chapter. In this final chapter, we present our final remarks, where we discuss the possible future directions to this research and present our conclusions.

13.1 Future Work

All the previous chapters are the result of a maturing field. The research and the principles reported in this document would not be possible without the accumulation of the prior work. In a similar manner, the research presented in this document is just another step in the SEE field that needs to be investigated and improved with further research. In this section we share our ideas regarding how our research can be extended.

Larger Data Sets: One of the possible future directions to the research presented in this thesis is the application of the proposed methods to larger data sets. In SEE, the data sets are usually limited to a couple of hundred instances at most. That is quite normal, given the fact that each instance of an SEE data set is a completed software project, which can take months or years to complete. However, small data sets is not always the case for other domains. The proposed methods in this document can also be adapted to big data problems. For example, currently we are applying the QUICK algorithm introduced in Chapter 11 on defect prediction data sets, which are orders of magnitude bigger compared to SEE data sets. Our initial results are quite promising, which shows that it is possible to extend the proposed methods to larger data sets. The problem inherent in applying these algorithms to larger data sets is related to implementation issues. Depending on the size of the data set, it may be necessary to find low-memory alternatives to implementation or to use parallelism in the programming of the proposed algorithms.

Change of Domains: A very promising future direction can be to extend the solutions proposed in this work to domains other than SEE. Note that although the presented solutions are developed to solve issues associated with SEE domain, the addressed issues are common in other domains. For example, finding an essential content of a noisy data set can be beneficial for different domains. One can use such a solution to summarize a large and noisy data set, so that the experts can *look* at the core of the data manually and interpret it. Another example can be the use of filtering to facilitate cross data usage. The lack of local data is not necessarily restricted to SEE domain and the proposed solutions of filtering relevant cross data can be applied to different domains.

Smarter Algorithms: This thesis introduces multiple different algorithms for different kind of problems. For example, in Chapter 5 we see that it is possible to find a subset of more successful methods with less rank deltas (so called superior solo-methods); in Chapter 6 we make use of the superior solo-methods to come up with even better performing multi-methods. However, the process of coming up with superior solo-methods require trial of a wide range of pre-processors and learners. It should be possible to make this process faster and smarter, also in an automated manner. In other words, the process to find superior solo-methods can be investigated as a future direction to come up with a quicker way of finding them. Which in return will make the formation of ensembles more straightforward. Note that formation of multi-methods use simple means of combining solo-methods. So, an interesting direction would be to investigate smarter means of combining solo-methods. Also note that multi-methods performed better than almost all the solo-methods in a big majority of the cases, which shows that it is possible to improve the performance of the solo-methods. Another future direction could be to compare how the solo and multi-methods differ in terms of their prediction at the instance level and improve solo-methods accordingly. This future direction could also lead the gateway to even more successful multi-methods.

Improving Common Ideas: Note that Chapter 8, Chapter 9, Chapter 10 and Chapter 11 all have common properties. Although the proposed algorithms of D-ABE, TEAK, pop1NN and QUICK work in different ways, they all share the fundamental property of getting rid of the unnecessary parts of the data. D-ABE, TEAK, pop1NN deal with the row pruning, whereas QUICK remarkably shows that row and column pruning can be combined to come up with a much smaller essential content of the SEE data sets. However, note that all these methods work based on nearest-neighbor ideas, in other words, if implementation of these methods are not done with performance considerations they may turn out to be taking too much execution time. Caching the distances between all the instances is a trick that we used during the implementation. However, there may be ways other than nearest-neighbor calculations to prune away instances and features in SEE data sets. Considering different alternatives to identify which rows and features to prune may be a good future direction. For example, an obvious future direction could be to investigate linear time nearest-neighbor methods, which would dramatically decrease the run times. Another future direction could be to use other alternatives to point out irrelevant instances and features, e.g. the variance or entropy of the features (to prune away features); projection of features to lower dimensions so as to see which instances cluster together and which features stay as outliers and so on.

Much Data to Touch: The fundamental aim of this thesis is to make a contribution to the SEE domain. The previous future work directions point out the facts that the presented algorithms: 1) Can be used for larger data sets; 2) Can be extended to other domains; 3) Can be improved to work in a smarter way; 4) Can be made more efficient. All these suggestions can hint a researcher to get involved with the ever increasing public data sets. For example, social media presents a giant pile of structured or unstructured data. Dealing with such big data sources as a future work and adapting the proposed ideas and solutions should inevitably include all or part of the the prior future directions. Therefore, an extension of this work as a future direction could be to tackle the publicly accessible big data sources and their problems. Note that the pursuit of big data problems with the proposed solutions is likely to call not only for prediction purposes, but also for the reasoning regarding the investigated data. Solid reasoning about the big data sources in or to provide trade-off ideas, future plans and summarizations of the observed phenomena will surely spur further algorithms and ideas.

13.2 Conclusion

The research presented in this document is built on the prior SEE work of more than 3 decades. The accumulation of the prior work identifies the important problems of SEE. The questions such as: “What is the best SEE method?”; “How can we handle the lack of local data?”; “How can we avoid the use of size attributes?”; “How can find an essential subset of the SEE data sets?”; “How should we choose sampling methods for SEE experiments?” are still open and we lack concrete answers to these questions. These questions are critical for the issues of new SEE methods, the understanding of the SEE data sets, experimentation as well as the sharing of experiments. The research presented in this document tackles the important and open questions of SEE; hence, it provides an important insight to the critical issues of SEE methods, data sets and experimentation.

The afore-mentioned issues are strongly related to industry practices, which often require a researcher/practitioner to deal with data collection in industrial settings. During the duration of this thesis, we have dealt with industrial projects and it is our opinion that sharing of our experiences can prove to be helpful. Therefore, the first 4 principles are the result of our experiences and are less technical than the following chapters. These 4 principles are:

1) Know your domain
2) Let the Experts Talk
3) Suspect your data
4) Data Collection is Cyclic

We see the contributions of our discussion regarding the data collection experiences as follows:

- Sharing of hands-on experience with industrial data collection
- Identification of possible pitfalls and likely solutions

In the remainder of the chapters we summarize the principles that emerged as a rigorous experimentation with a large corpus of SEE methods and data sets. These chapters target finding successful SEE methods with small rank changes; providing methods to evaluate SEE methods; improving ABE methods; targeting the issues of cross domain data; compensating the lack of size features; finding an essential set of SEE data sets and providing an evaluation of the sampling methods. The principles that emerged as a result of this extensive experimentation on the critical issues of SEE can be listed as follows:

5) Use a Ranking Stability Indicator
6) Assemble Superior Methods
7) Weighting Analogies is Over-elaboration
8) Use Easy-path Design

9) Use Relevancy Filtering
10) Use Outlier Pruning
11) Combine Outlier and Synonym Pruning
12) Be Aware of Sampling Method Trade-off

There exists a number of contributions associated with the research that led to each of these principles. The summary of all the contributions can be listed as follows:

- A method to identify successful methods using their rank changes.
- An evaluation method of the diversity of the SEE data sets.
- A novel scheme for assembling the best solo-methods.
- Stable multi-methods that outperform solo-methods.
- Investigation of an unexplored and promising ABE option of kernel-weighting, which reduces the ABE variants space to be explored by 2090 ABE scenarios
- An ABE design principle (easy-path) applicable to different ABE methods.
- Discovering the performance between *static-k* based ABE methods and TMPA.
- Utilization of a novel ABE method for time interval transfer.
- Successful transfer learning method on proprietary as well as public data sets.
- Promotion of SEE methods that can compensate the lack of size features.
- A method called pop1NN that shows that size features are not a "must".
- An unsupervised method to find the essential content of SEE data sets.
- Promoting research to elaborate on the data, not on the algorithm.
- The first systematic investigation of *B&V* trade-off in SEE domain.
- Showing that *B&V* trade-off and run times are not the main concerns for SMs
- Recommendation of LOO as it avoids non-deterministic of train and test sets.

References

[1] E. Mendes, I. D. Watson, C. Triggs, N. Mosley, and S. Counsell, “A comparative study of cost estimation models for web hypermedia applications,” *Empirical Software Engineering*, vol. 8, no. 2, pp. 163–196, 2003.

[2] E. Kocaguneli and T. Menzies, “How to find relevant data for effort estimation,” in *ESEM’11: International Symposium on Empirical Software Engineering and Measurement*, 2011.

[3] M. Jorgensen and M. Shepperd, “A systematic review of software development cost estimation studies,” *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 33–53, 2007.

[4] M. Shepperd and G. Kadoda, “Comparing Software Prediction Techniques Using Simulation,” *IEEE Trans. Softw. Eng.*, vol. 27, no. 11, pp. 1014–1022, 2001. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=965341

[5] T. Menzies, C. Bird, T. Zimmermann, W. Schulte, and E. Kocaganeli, “The inductive software engineering manifesto: principles for industrial data mining,” in *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, ser. MALETS ’11, 2011, pp. 19–26.

[6] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas, “On combining classifiers,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 226–239, 1998.

[7] K. Ali, “On the link between error correlation and error reduction in decision tree ensembles,” Dept. of Information and Computer Science, Univ. of California, Irvine, Tech. Rep., 1995.

[8] E. Alpaydin, “Techniques for combining multiple learners,” *Proceedings of Engineering of Intelligent Systems*, vol. 2, pp. 6–12, 1998.

[9] T. M. Khoshgoftaar, P. Rebours, and N. Seliya, “Software quality analysis by combining multiple projects and learners,” *Software Quality Control*, vol. 17, no. 1, pp. 25–49, 2009.

[10] E. Kocaguneli, Y. Kultur, and A. Bener, “Combining multiple learners induced on multiple datasets for software effort prediction,” in *International Symposium on Software Reliability Engineering (ISSRE)*, 2009, student Paper.

[11] M. C. K. Vinaykumar, V. Ravi, “Software cost estimation using soft computing approaches,” *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pp. 499–518, 2010.

[12] E. Mendes and N. Mosley, “Bayesian network models for web effort prediction: A comparative study,” *IEEE Trans. Softw. Eng.*, vol. 34, pp. 723–737, 2008.

[13] J. W. Keung, B. A. Kitchenham, and D. R. Jeffery, “Analogy-x: Providing statistical inference to analogy-based software cost estimation,” *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 471–484, 2008.

[14] I. Myrtveit, E. Stensrud, and M. Shepperd, “Reliability and validity in comparative studies of software prediction models,” *IEEE Trans. Softw. Eng.*, vol. 31, no. 5, pp. 380–391, May 2005.

[15] F. Walkerden and R. Jeffery, “An empirical study of analogy-based software effort estimation,” *Empirical Software Engineering*, vol. 4, no. 2, pp. 135–158, 1999.

[16] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung, “Exploiting the essential assumptions of analogy-based effort estimation,” *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 425 –438, march-april 2012.

[17] E. Mendes and N. Mosley, “Further investigation into the use of cbr and stepwise regression to predict development effort for web hypermedia applications,” in *International Symposium on Empirical Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2002.

[18] E. Kocaguneli, J. Keung, and T. Menzies, “Towards theoretical maximum prediction accuracy using d-abe,” *submitted to IEEE. Trans. on Softw. Eng.*, 2012.

[19] E. Kocaguneli, G. Gay, Y. Yang, T. Menzies, and J. Keung, “When to use data from other projects for effort estimation,” in *ASE ’10: To Appear In the Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, New York, NY, USA, 2010.

[20] B. W. Boehm, *Software Engineering Economics*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[21] B. A. Kitchenham, E. Mendes, and G. H. Travassos, “Cross versus within-company cost estimation studies: A systematic review,” *IEEE Trans. Softw. Eng.*, vol. 33, no. 5, pp. 316–329, 2007.

[22] B. Turhan, T. Menzies, A. Bener, and J. Di Stefano, “On the relative value of cross-company and within-company data for defect prediction,” *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009. [Online]. Available: http://www.springerlink.com/index/46U3115855511337.pdf

[23] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, “Cross-project defect prediction: A large scale experiment on data vs. domain vs. process,” *ESEC/FSE*, pp. 91–100, 2009.

[24] J. W. Keung, “Theoretical Maximum Prediction Accuracy for Analogy-Based Software Cost Estimation,” *15th Asia-Pacific Software Engineering Conference*, pp. 495–502, 2008. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4724583

[25] L. Briand and I. Wieczorek, *Resource Modeling in Software Engineering*, 2nd ed., ser. Encyclopedia of Software Engineering. Wiley, 2002.

[26] T. Menzies, Z. Chen, J. Hihn, and K. Lum, “Selecting best practices for effort estimation,” *IEEE Trans. Softw. Eng.*, vol. 32, pp. 883–895, 2006.

[27] M. Jorgensen, “A review of studies on expert estimation of software development effort,” *Journal of Systems and Software*, vol. 70, pp. 37–60, 2004.

[28] M. Shepperd, C. Schofield, and B. Kitchenham, “Effort estimation using analogy,” in *ICSE ’96: Proceedings of the 18th international conference on Software engineering*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 170–178.

[29] G. Rowe and G. Wright, “The delphi technique as a forecasting tool: issues and analysis,” *International Journal of Forecasting*, vol. 15, 1999.

[30] K. Atkinson and M. Shepperd, “The use of function points to find cost analogies,” *European Software Cost Modelling Meeting . Ivrea, Italy*, 1994.

[31] M. Jorgensen, “Practical guidelines for expert-judgment-based software effort estimation,” *IEEE Softw.*, vol. 22, no. 3, pp. 57–63, 2005.

[32] A. Bakir, B. Turhan, and A. B. Bener, “A new perspective on data homogeneity in software cost estimation: a study in the embedded systems domain,” *Software Quality Control*, vol. 18, pp. 57–80, March 2010. [Online]. Available: http://dx.doi.org/10.1007/s11219-009-9081-z

[33] A. Albrecht and J. Gaffney, “Software function, source lines of code and development effort prediction: A software science validation,” *IEEE Transactions on Software Engineering*, vol. 9, pp. 639–648, 1983.

[34] B. Kitchenham and K. Känsälä, “Inter-item correlations among function points,” in *ICSE’93:Proceedings of the 15th international Conference on Software Engineering*, ser. ICSE ’93. Los Alamitos, CA, USA: IEEE Computer Society Press, 1993, pp. 477–480. [Online]. Available: http://dl.acm.org/citation.cfm?id=257572.257677

[35] C. Kemerer, “An empirical validation of software cost estimation models,” *Communications of the ACM*, vol. 30, no. 5, pp. 416–429, May 1987.

[36] K. D. Maxwell, *Applied Statistics for Software Managers*. Prentice Hall, PTR, 2002.

[37] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki, “Robust regression for developing software estimation models,” *J. Syst. Softw.*, vol. 27, no. 1, pp. 3–16, 1994.

[38] M. Shepperd and C. Schofield, “Estimating software project effort using analogies,” *IEEE Trans. Softw. Eng.*, vol. 23, no. 11, pp. 736–743, 1997.

[39] E. Mendes, N. Mosley, and S. Counsell, “Investigating web size metrics for early web cost estimation,” *The Journal of Systems and Software*, vol. 77, pp. 157–172, August 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2004.08.034

[40] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, “How effective is tabu search to configure support vector regression for effort estimation?” in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, 2010.

[41] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, “A simulation study of the model evaluation criterion mmre,” *IEEE Trans. Softw. Eng.*, vol. 29, no. 11, pp. 985–995, 2003.

[42] M. Shepperd and S. Macdonell, “Evaluating Prediction Systems in Software Project Estimation,” *Information and Software Technology*, vol. pre-prints, pp. 1–21, 2011.

[43] B. Kitchenham and E. Mendes, “Why comparative effort prediction studies may be invalid,” in *PROMISE ’09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. New York, NY, USA: ACM, 2009, pp. 1–5.

[44] J. Kliijnen, “Sensitivity analysis and related analyses: a survey of statistical techniques,” *Journal Statistical Computation and Simulation*, vol. 57, no. 1–4, pp. 111–142, 1997.

[45] E. Alpaydin, *Introduction to Machine Learning*. MIT Press, 2004.

[46] Y. Li, M. Xie, and T. Goh, “A study of project selection and feature weighting for analogy based software cost estimation,” *Journal of Systems and Software*, vol. 82, pp. 241–252, 2009.

[47] G. Kadoda, M. Cartwright, and M. Shepperd, “On configuring a case-based reasoning software project prediction system,” *UK CBR Workshop, Cambridge, UK*, pp. 1–10, 2000.

[48] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[49] J. Keung, E. Kocaguneli, and T. Menzies, “A Ranking Stability Indicator for Selecting the Best Effort Estimator in Software Cost Estimation,” *Automated Software Engineering, to appear in 2012)*, 2011. [Online]. Available: http://menzies.us/pdf/11draftranking.pdf

[50] J. Keung and T. Nguyen, “Quantitative Analysis for Non-linear System Performance Data Using Case-Based Reasoning,” in *Asia Pacific Software Engineering Conference*, 2010, pp. 346–355. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/APSEC.2010.47

[51] A. Aamodt and E. Plaza, "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," *Artificial Intellegence Communications*, vol. 7, pp. 39–59, 1994.

[52] S. K. Pal and S. C. K. Shiu, *Foundations of Soft Case-based Reasoning*. Cambridge Univ Press, 2001.

[53] S. Craw, N. Wiratunga, and R. Rowe, "Learning adaptation knowledge to improve case-based reasoning," *Artificial Intelligence*, vol. 170, no. 16-17, pp. 1175–1192, Nov. 2006. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0004370206000798

[54] H. Li, D. Hu, T. Hao, L. Wenyin, and X. Chen, "Adaptation Rule Learning for Case-Based Reasoning," *Third International Conference on Semantics, Knowledge and Grid (SKG 2007)*, pp. 44–49, Oct. 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4438508

[55] C.-L. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Trans. on Computers*, vol. C-23, no. 11, pp. 1179 – 1184, Nov 1974.

[56] M. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions On Knowledge And Data Engineering*, vol. 15, no. 6, pp. 1437–1447, 2003.

[57] U. M. Fayyad and I. H. Irani, "Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning," in *Proceedings of the International Joint Conference on Uncertainty in AI*, 1993, pp. 1022–1027.

[58] Y. Yang and G. I. Webb, "A Comparative Study of Discretization Methods for Naive-Bayes Classifiers," in *PKAW '02: The Pacific Rim Knowledge Acquisition Workshop*, 2002, pp. 159–173.

[59] E. Frank, M. Hall, and B. Pfahringer, "Locally weighted naive bayes," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 2003, pp. 249–256.

[60] L. Angelis and I. Stamelos, "A simulation tool for efficient analogy based cost estimation," *Empirical Softw. Eng.*, vol. 5, pp. 35–68, 2000.

[61] D. 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.

[62] C. Robson, "Real world research: a resource for social scientists and practitioner-researchers," *Blackwell Publisher Ltd*, 2002.

[63] D. Milicic and C. Wohlin, "Distribution patterns of effort estimations," in *EUROMICRO*, 2004, pp. 422–429.

[64] B. Kitchenham, E. Mendes, and G. H. Travassos, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Trans. Softw. Eng.*, vol. 33, no. 5, pp. 316–329, 2007.

[65] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The kdd process for extracting useful knowledge from volumes of data," *Commun. ACM*, vol. 39, no. 11, pp. 27–34, Nov. 1996. [Online]. Available: http://doi.acm.org/10.1145/240455.240464

[66] J. Rauser, "What is a career in big data?" 2011. [Online]. Available: http://strataconf.com/stratany2011/public/schedule/speaker/10070

[67] T. Menzies, O. Jalali, J. Hihn, D. Baker, and K. Lum, "Stable rankings for different effort models," *Automated Software Engineering*, vol. 17, pp. 409–437, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10515-010-0070-z

[68] J. Li and G. Ruhe, "Decision support analysis for software effort estimation by analogy," in *International Conference on Predictive Models in Software Engineering PROMISE'07*, May 2007.

[69] L. Breiman, "Technical note: Some properties of splitting criteria," *Machine Learning*, vol. 24, pp. 41–47, 1996, 10.1023/A:1018094028462. [Online]. Available: http://dx.doi.org/10.1023/A:1018094028462

[70] M. Auer, A. Trendowicz, B. Graser, E. Haunschmid, and S. Biffl, "Optimal project feature weights in analogy-based cost estimation: Improvement and limitations," *IEEE Trans. Softw. Eng.*, vol. 32, pp. 83–92, 2006.

[71] C. Kirsopp, M. Shepperd, and R. House, "Case and feature subset selection in case-based software project effort prediction," *Research and development in intelligent systems XIX: proceedings of ES2002, the twenty-second SGAI International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, p. 61, 2003. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Case+and+Feature+Subset+Selection+in+Case-+Based+Software+Project+Effort+Prediction#0

[72] J. Li and G. Ruhe, "A comparative study of attribute weighting heuristics for effort estimation by analogy," *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, p. 74, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1159733.1159746

[73] J. Keung, "Empirical evaluation of analogy-x for software cost estimation," in *ESEM '08: Proceedings of the Second International Symposium on Empirical Software Engineering and Measurement*. New York, NY, USA: ACM, 2008, pp. 294–296.

[74] J. Keung and B. Kitchenham, "Experiments with analogy-x for software cost estimation," in *ASWEC '08: Proceedings of the 19th Australian Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 229–238.

[75] A. Brady and T. Menzies, “Case-based reasoning vs parametric models for software quality optimization,” in *International Conference on Predictive Models in Software Engineering PROMISE’10*. IEEE, Sept. 2010.

[76] G. Seni and J. Elder, *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*. Morgan and Claypool Publishers, 2010.

[77] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*, 2nd ed. Springer, 2008.

[78] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, San Francisco, CA, USA, 1995, pp. 1137–1143.

[79] L. Breiman and P. Spector, “Submodel selection and evaluation in regression. the x-random case,” *International Statistical Review*, vol. 60, no. 3, pp. 291–319, Decemeber 1992.

[80] T. G. Dietterich, “Ensemble methods in machine learning,” pp. 1–15, 2000.

[81] T. K. Ho, J. J. Hull, and S. N. Srihari, “Decision combination in multiple classifier systems,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 1, pp. 66–75, 1994.

[82] S. Günter and H. Bunke, “Feature selection algorithms for the generation of multiple classifier systems and their application to handwritten word recognition,” *Pattern Recogn. Lett.*, vol. 25, no. 11, pp. 1323–1336, 2004.

[83] T. K. Ho, “Random decision forests,” in *ICDAR ’95: Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1)*. Washington, DC, USA: IEEE Computer Society, 1995, p. 278.

[84] H. Zhao and S. Ram, “Constrained cascade generalization of decision trees,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, pp. 727–739, 2004.

[85] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementation*. Morgan Kaufmann, 2000.

[86] E. Bauer and R. Kohavi, “An empirical comparison of voting classification algorithms: Bagging, boosting, and variants,” *Machine Learning*, vol. 36, pp. 105–139, 1999, 10.1023/A:1007515423169. [Online]. Available: http://dx.doi.org/10.1023/A:1007515423169

[87] a. Ross, “Information fusion in biometrics,” *Pattern Recognition Letters*, vol. 24, no. 13, pp. 2115–2125, Sep. 2003. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167865503000795

[88] A. a. Ross, “Feature level fusion of hand and face biometrics,” *Proceedings of SPIE*, vol. 5779, no. March, pp. 196–204, 2005. [Online]. Available: http://link.aip.org/link/?PSI/5779/196/1&Agg=doi

[89] J. Pahariya, V. Ravi, and M. Carr, "Software cost estimation using computational intelligence techniques," *NaBIC'09: World Congress on Nature Biologically Inspired Computing*, pp. 849 –854, Dec. 2009.

[90] Y. Kultur, B. Turhan, and A. B. Bener, "ENNA: software effort estimation using ensemble of neural networks with associative memory," in *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, New York, NY, USA, 2008, pp. 330–338.

[91] B. Twala, M. Cartwright, and M. Shepperd, "Ensemble of missing data techniques to improve software prediction accuracy," in *ICSE '06: Proceedings of the 28th international Conference on Software engineering*. New York, NY, USA: ACM, 2006, pp. 909–912. [Online]. Available: http://doi.acm.org/10.1145/1134285.1134449

[92] T. M. Khoshgoftaar, S. Zhong, and V. Joshi, "Enhancing software quality estimation using ensemble-classifier based noise filtering," *Intell. Data Anal.*, vol. 9, pp. 3–27, January 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1239046.1239048

[93] Y. Freund and R. Schapire, "A decision-theoretic generalization of on- line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, 1997.

[94] Y. Jiang, B. Cukic, and T. Menzies, "Cost curve evaluation of fault prediction models," in *ISSRE '08: Proceedings of the 2008 19th International Symposium on Software Reliability Engineering*, 2008, pp. 197–206.

[95] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[96] R. M. Hogarth and H. Kunreuther, "Pricing insurance and warranties: Ambiguity and correlated risks," *The GENEVA Papers on Risk and Insurance - Theory*, vol. 17, no. 1, pp. 35–60, 1992.

[97] A. Oppeneheim, A. Wilsky, and S. Hamid, *Signals and Systems*. Prentice Hall, 1996.

[98] A. Nelson, T. Menzies, and G. Gay, "Sharing experiments using open-source software," *Software: Practice and Experience*, vol. 41, no. 3, pp. 283–305, 2011.

[99] A. Bakir, E. Kocaguneli, A. Tosun, A. Bener, and B. Turhan, "Xiruxe: An Intelligent Fault Tracking Tool," in *AIPR'09: International Conference on Artificial Intelligence and Pattern Recognition*, Orlando, 2009.

[100] A. Tosun, A. Bener, and E. Kocaguneli, "BITS: Issue Tracking and Project Management Tool in Healthcare Software Development," in *SEKE'09: International Conference on Software Engineering and Knowledge Engineering*, Boston, MA, USA, 2009.

[101] E. Kocaguneli, A. Tosun, A. Bener, B. Caglayan, and B. Turhan, "Prest: An Intelligent Software Metrics Extraction, Analysis and Defect Prediction Tool," in *SEKE'09: International Conference on Software Engineering and Knowledge Engineering*, Boston, 2009.

[102] J. S. Armstrong, “Significance tests harm progress in forecasting,” *International Journal of Forecasting*, vol. 23, no. 2, pp. 321–327, 2007.

[103] ——, *Principles of Forecasting: A Handbook for Researchers and Practitioners*. Kluwer Academic, 2001.

[104] R. M. Hogarth, “A note on aggregating opinions,” *Organizational Behavior and Human Performance*, vol. 21, no. 1, pp. 40 – 46, 1978.

[105] J. Li and G. Ruhe, “Analysis of attribute weighting heuristics for analogy-based software effort estimation method AQUA+,” *Empirical Software Engineering*, vol. 13, no. 1, pp. 63–96, 2008. [Online]. Available: http://www.springerlink.com/index/P821335293V53481.pdf

[106] J. Li, G. Ruhe, A. Al-emran, and M. M. Richter, “A flexible method for software effort estimation by analogy,” *Empirical Software Engineering*, vol. 12, pp. 65–106, 2007.

[107] T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, “Distributed deviation detection in sensor networks,” *SIGMOD Rec*, vol. 32, p. 2003, 2003.

[108] G. John and P. Langley, “Estimating continuous distributions in bayesian classifiers,” in *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1995, pp. 338–345.

[109] N. A. C. Cressie, *Statistics for Spatial Data (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 1993. [Online]. Available: http://www.worldcat.org/isbn/0471002550

[110] W. Hardle and L. Simar, *Applied Multivariate Statistical Analysis*. Springer Verlag, 2003.

[111] L. C. Briand, K. El Emam, D. Surmann, I. Wieczorek, and K. D. Maxwell, “An assessment and comparison of common software cost estimation modeling techniques,” in *ICSE ’99: Proceedings of the 21st international conference on Software engineering*. New York, NY, USA: ACM, 1999, pp. 313–322.

[112] R. Jeffery, M. Ruhe, and I. Wieczorek, “Using public domain metrics to estimate software development effort,” in *METRICS ’01: Proceedings of the 7th International Symposium on Software Metrics*. Washington, DC, USA: IEEE Computer Society, 2001, p. 16.

[113] H. I. Browman, “Negative results,” *Mar. Ecol. Prog. Ser.*, vol. 191, pp. 301–309, 1999. [Online]. Available: http://www.springerlink.com/index/N7K33668Q00G5105.pdf

[114] S. Scheid, “Introduction to kernel smoothing,” Talk, January 2004.

[115] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd Edition)*, 2nd ed. Wiley-Interscience, Nov. 2001. [Online]. Available: http://www.worldcat.org/isbn/0471056693

[116] M. P. Wand and M. C. Jones, *Kernel Smoothing (Monographs on Statistics and Applied Probability)*. Chapman & Hall/CRC, December 1994. [Online]. Available: http://www.worldcat.org/isbn/0412552701

[117] D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization (Wiley Series in Probability and Statistics)*. Wiley-Interscience, September 1992.

[118] J. R. Quinlan, "Boosting first-order learning," in *Proceedings of the 7th International Workshop on Algorithmic Learning Theory*, vol. 1160, 1996, pp. 143–155.

[119] U. Lipowezky, "Selection of the optimal prototype subset for 1-nn classification," *Pattern Recognition Letters*, vol. 19, pp. 907–918, 1998.

[120] C. Kirsopp, M. Shepperd, and R. Premrag, "Case and feature subset selection in case-based software project effort prediction," *Research and development in intelligent systems XIX: proceedings of ES2002, the twenty-second SGAI International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, p. 61, 2002.

[121] N. Segata, E. Blanzieri, S. J. Delany, and P. Cunningham, "Noise reduction for instance-based learning with a local maximal margin approach," *Journal of Intelligent Information Systems*, vol. 35, no. 2, pp. 301–331, Aug. 2009. [Online]. Available: http://www.springerlink.com/index/10.1007/s10844-009-0101-z

[122] K. Lum, T. Menzies, and D. Baker, "2cee, a twenty first century effort estimation methodology," in *International Society of Parametric Analysis Conference (ISPA / SCEA)*, May 2008.

[123] T. K. Le-Do, K.-A. Yoon, Y.-S. Seo, and D.-H. Bae, "Filtering of inconsistent software project data for analogy-based effort estimation," *Annual International Computer Software and Applications Conference*, pp. 503–508, 2010.

[124] T. Khoshgoftaar, L. Bullard, and K. Gao, "Detecting outliers using rule-based modeling for improving cbr-based software quality classification models," in *Case-Based Reasoning Research and Development*, ser. Lecture Notes in Computer Science. Springer Berlin - Heidelberg, 2003, vol. 2689, pp. 1063–1063.

[125] D. Wilson and T. Martinez, "Reduction techniques for instance-based learning algorithms," *Machine Learning*, vol. 38, no. 3, pp. 257–286, 2000. [Online]. Available: http://www.springerlink.com/index/n748p82037443824.pdf

[126] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data Min. Knowl. Discov.*, vol. 6, pp. 153–172, April 2002. [Online]. Available: http://portal.acm.org/citation.cfm?id=593433.593527

[127] V. Nguyen, L. Huang, and B. Boehm, "An analysis of trends in productivity and cost drivers over years," in *Promise '11: Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, 2011, pp. 3:1–3:10.

[128] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345 –1359, 2010.

[129] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248 – 256, 2012.

[130] A. Arnold, R. Nallapati, and W. Cohen, “A comparative study of methods for transductive transfer learning,” in *ICDM’07: Seventh IEEE International Conference on Data Mining Workshops*, 2007, pp. 77 –82.

[131] G. Foster, C. Goutte, and R. Kuhn, “Discriminative instance weighting for domain adaptation in statistical machine translation,” in *EMNLP ’10: Conference on Empirical Methods in Natural Language Processing*, 2010, pp. 451–459.

[132] S.-I. Lee, V. Chatalbashev, D. Vickrey, and D. Koller, “Learning a meta-level prior for feature relevance from multiple related tasks,” in *ICML ’07: Proceedings of the 24th international conference on Machine learning*, 2007, pp. 489–496.

[133] J. Gao, W. Fan, J. Jiang, and J. Han, “Knowledge transfer via multiple model local structure mapping,” in *In International Conference on Knowledge Discovery and Data Mining, Las Vegas, NV*, 2008.

[134] L. Mihalkova, T. Huynh, and R. J. Mooney, “Mapping and revising markov logic networks for transfer learning,” in *AAAI’07: Proceedings of the 22nd national conference on Artificial intelligence*, 2007, pp. 608–614.

[135] W. Dai, G.-R. Xue, Yang, Qiang, and Y. Yong, “Transferring naive bayes classifiers for text classification,” in *AAAI’07: Proceedings of the 22nd national conference on Artificial intelligence*, 2007, pp. 540–545.

[136] X. Zhang, W. Dai, G.-R. Xue, and Y. Yu, “Adaptive email spam filtering based on information theory,” in *Web Information Systems Engineering WISE 2007*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4831, pp. 159–170.

[137] P. Wu and T. G. Dietterich, “Improving svm accuracy by training on auxiliary data sources,” in *Proceedings of the twenty-first international conference on Machine learning*, ser. ICML ’04. New York, NY, USA: ACM, 2004, pp. 110–. [Online]. Available: http://doi.acm.org/10.1145/1015330.1015436

[138] C. Lokan and E. Mendes, “Using chronological splitting to compare cross- and single-company effort models: further investigation,” in *Proceedings of the Thirty-Second Australasian Conference on Computer Science - Volume 91*, ser. ACSC ’09, 2009, pp. 47–54.

[139] ——, “Applying moving windows to software effort estimation,” in *ESEM’09: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 111–122. [Online]. Available: http://dx.doi.org/10.1109/ESEM.2009.5316019

[140] J. Huang, A. Smola, A. Gretton, K. Borgwardt, and B. Scholkopf, “Correcting sample selection bias by unlabeled data,” in *Proceedings of the 19th Annual Conference on Neural Information Processing Systems*, 2007, pp. 601–608.

[141] C.-l. Chang, “Finding Prototypes for Nearest Classifiers,” *IEEE Transactions on Computer*, vol. C, no. 11, 1974.

[142] K. Gollapudi, “Function points or lines of code? - an insight,” in *Global Microsoft Business Unit, Wipro Technologies*, 2004.

[143] E. W. Dijkstra, “On the cruelty of really teaching computing science,” 1988. [Online]. Available: http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1036.PDF

[144] B. Boehm, “Safe and Simple Software Cost Analysis,” *IEEE Software*, pp. 14–17, 2000.

[145] J. Desharnais, “Analyse statistique de la productivitie des projets informatique a partie de la technique des point des fonction,” Master’s thesis, Univ. of Montreal, 1989.

[146] Y. Li, M. Xie, and T. Goh, “A study of mutual information based feature selection for case based reasoning in software cost estimation,” *Expert Systems with Applications*, vol. 36, no. 3, pp. 5921 – 5931, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417408004429

[147] D. Posnett, V. Filkov, and P. Devanbu, “Ecological inference in empirical software engineering,” in *ASE’11: International Conference on Automated Software Engineering*, 2011, pp. 362–371.

[148] E. Kocaguneli, T. Menzies, and J. Keung, “On the value of ensemble effort estimation,” *IEEE Trans. Softw. Eng.*, vol. PP, no. 99, p. 1, 2011.

[149] K. Petersen and C. Wohlin, “Context in industrial software engineering research,” in *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, Oct. 2009, pp. 401 –404.

[150] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, “Local vs global models for effort estimation and defect prediction,” in *IEEE ASE’11*, 2011, available from http://menzies.us/pdf/11ase.pdf.

[151] C. Passos, A. P. Braun, D. S. Cruzes, and M. Mendonca, “Analyzing the impact of beliefs in software project practices,” in *ESEM’11*, 2011.

[152] M. Azzeh, “Software effort estimation based on optimized model tree,” in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, ser. Promise ’11. New York, NY, USA: ACM, 2011, pp. 6:1–6:8. [Online]. Available: http://doi.acm.org/10.1145/2020390.2020396

[153] M. Korte and D. Port, “Confidence in software cost estimation results based on mmre and pred,” in *Proceedings of the 4th international workshop on Predictor models in software engineering*, ser. PROMISE ’08. New York, NY, USA: ACM, 2008, pp. 63–70. [Online]. Available: http://doi.acm.org/10.1145/1370788.1370804

[154] N. Al Khalidi, A. Saifan, and I. Alsmadi, “Selecting a standard set of attributes for cost estimation of software projects,” in *Computer, Information and Telecommunication Systems (CITS), 2012 International Conference on*, may 2012, pp. 1 –5.

[155] P. Jodpimai, P. Sophatsathit, and C. Lursinsap, "Estimating software effort with minimum features using neural functional approximation," in *Computational Science and Its Applications (ICCSA), 2010 International Conference on*, march 2010, pp. 266 –273.

[156] S. Arun Kumar and T. Arun Kumar, "State of software metrics to forecast variety of elements in the software development process," in *Advances in Parallel Distributed Computing*, ser. Communications in Computer and Information Science, D. Nagamalai, E. Renault, and M. Dhanuskodi, Eds. Springer Berlin Heidelberg, 2011, vol. 203, pp. 561–569.

[157] Y. Li, M. Xie, and G. T., "A study of the non-linear adjustment for analogy based software cost estimation," *Empirical Software Engineering*, pp. 603–643, 2009.

[158] P. Sentas, L. Angelis, I. Stamelos, and G. Bleris, "Software productivity and effort prediction with ordinal regression," *Information and Software Technology*, vol. 47, no. 1, pp. 17 – 29, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0B-4CVR4MP-1/2/72e600ea322bfe4d64b1c90c3a968013

[159] G. R. Finnie, G. E. Wittig, and J.-M. Desharnais, "A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models," *Journal of Systems and Software*, vol. 39, no. 3, pp. 281 – 289, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0N-3SP2RBC-6/2/4416e483f8e9fa78f01486d8fa6b7693

[160] A. L. Oliveira, P. L. Braga, R. M. Lima, and M. L. Cornalio, "Ga-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation," *Information and Software Technology*, vol. 52, no. 11, pp. 1155 – 1166, 2010, ¡ce:title¿Special Section on Best Papers PROMISE 2009¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584910000984

[161] S.-B. Roh, S.-K. Oh, and W. Pedrycz, "Design of fuzzy radial basis function-based polynomial neural networks," *Fuzzy Sets and Systems*, vol. 185, no. 1, pp. 15 – 37, 2011, ¡ce:title¿Theme: Systems Engineering¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165011411003009

[162] B. Turhan, O. Kutlubay, and A. Bener, "Evaluation of feature extraction methods on software cost estimation," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 20-21 2007, pp. 497 –497.

[163] M. Shepperd, "It doesn't matter what you do but does matter who does it!" in *CREST Open Workshop, University College, October 24-25*, 2011.

[164] K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens, "Data mining techniques for software effort estimation: A comparative study," *IEEE Trans. on Softw. Eng.*, vol. PP, no. 99, p. 1, 2011.

[165] S. Dasgupta, "Analysis of a greedy active learning strategy," *in Neural Information Processing Systems 17:*, vol. 1, no. x, 2005.

[166] B. Wallace, K. Small, C. Brodley, and T. Trikalinos, “Active learning for biomedical citation screening,” in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 173–182. [Online]. Available: http://portal.acm.org/citation.cfm?id=1835829

[167] J. F. Bowring, J. M. Rehg, and M. J. Harrold, “Active learning for automatic classification of software behavior,” *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 4, p. 195, Jul. 2004. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1013886.1007539

[168] T. Xie and D. Notkin, “Mutually enhancing test generation and specification inference,” *Formal Approaches to Software Testing*, pp. 1100–1101, 2004. [Online]. Available: http://www.springerlink.com/index/GJEU06J1KGQ95GYX.pdf

[169] A. Hassan and T. Xie, “Software intelligence: the future of mining software engineering data,” in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 161–166. [Online]. Available: http://portal.acm.org/citation.cfm?id=1882397

[170] Z. Chen, T. Menzies, and D. Port, “Feature Subset Selection Can Improve Software Cost Estimation,” in *PROMISE’05: Proceedings of the International Conference on Predictor Models in Software Engineering*, 2005.

[171] M. Azzeh, D. Neagu, and P. Cowling, “Improving analogy software effort estimation using fuzzy feature subset selection algorithm,” in *PROMISE ’08: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*. New York, NY, USA: ACM, 2008, pp. 71–78.

[172] A. Bakir, B. Turhan, and A. Bener, “A comparative study for estimating software development effort intervals,” *Software Quality Journal*, vol. 19, pp. 537–552, 2011.

[173] G. Kadoda, M. Cartwright, L. Chen, and M. Shepperd, “Experiences using case-based reasoning to predict software project effort,” in *EASE’00: Fourth International Conference on Empirical Assessment and Evaluation in Software Engineering*, 2000.

[174] R. Valerdi, “Heuristics for systems engineering cost estimation,” *IEEE Systems Journal*, vol. 5, no. 1, pp. 91–98, 2011.

[175] J. Demsar, “Statistical Comparisons of Clasifiers over Multiple Data Sets,” *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[176] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking classification models for software defect prediction: A proposed framework and novel findings,” *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485 –496, 2008.

[177] L. Briand, T. Langley, and I. Wieczorek, “A replicated assessment and comparison of common software cost modeling techniques,” *ICSE’00: Proceedings of the 22nd International Conference on Software Engineering*, pp. 377–386, 2000.

[178] ——, “European space agency database: An assessment of common software cost techniques,” International Software Engineering Research Network, Tech. Rep., 1999.

[179] A. M. Molinaro, R. Simon, and R. M. Pfeiffer, “Prediction error estimation: a comparison of resampling methods,” *Bioinformatics (Oxford, England)*, vol. 21, no. 15, pp. 3301–7, Aug. 2005.

[180] M. Hollander and D. A. Wolfe, *Nonparametric Statistical Methods*. Hoboken, NJ: John Wiley and Sons, Inc., 1999.

[181] Y. Hochberg and A. C. Tamhane, *Multiple Comparison Procedures*. Hoboken, NJ: John Wiley and Sons, Inc., 1987.