Size Doesn't Matter? On the Value of Software Size Features for Effort Estimation

Ekrem Kocaguneli[†], Tim Menzies[‡]
LCSEE
West Virginia University
Morgantown, USA
ekocagun@mix.wvu.edu[†], tim@menzies.us[‡]

Jairus Hihn
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, USA
jairus.hihn@jpl.nasa.gov

Byeong Ho Kang
School of Computing & Information Systems
University of Tasmania
Australia
byeong.kang@utas.edu.au

ABSTRACT

Background: Size features such as lines of code and function points are deemed essential for effort estimation. No one questions under what conditions size features are actually a "must".
Aim: To question the need for size features and to propose a method that compensates their absence.
Method: A baseline analogy-based estimation method (1NN) and a state-of-the-art learner (CART) are run on reduced (with no size features) and full (with *all* features) versions of 13 SEE data sets. 1NN is augmented with a popularity-based pre-processor to create *"pop1NN"*. The performance of pop1NN is compared to 1NN and CART using 10-way cross validation w.r.t. MMRE, MdMRE, MAR, PRED(25), MBRE, MIBRE, and MMER.
Results: Without any pre-processor, removal of size features decreases the performance of 1NN and CART. For 11 out of 13 data sets, pop1NN removes the necessity of size features. pop1NN (using reduced data) has a comparable performance to CART (using full data).
Conclusion: Size features are important and their use is endorsed. However, if there are insufficient means to collect software size metrics, then the use of methods like pop1NN may compensate for size metrics with only a small loss in estimation accuracy.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management — cost estimation

General Terms

Management, Measurement

Keywords

lines of code, function points, instance selection, popularity, analogy-based estimation, k-NN

PROMISE '12, September 21–22, 2012, Lund, Sweden

1. INTRODUCTION

Software effort estimation (SEE) is a maturing field of software engineering (SE). Yet even after decades of research and model development, SEE still remains a tricky task under the best of circumstances. There are extensive debates as to which methods are best, mostly focusing on model vs. analogy based methods. Shepperd et al. reports analogy-based estimation (ABE) methods as the high performing methods [1]; whereas within DOD and NASA, parametric effort models like COCOMO [2] are extensively used and have been shown to be high performing methods under the right conditions [3]. Hihn et al. have found that among practitioners ABE and expert-based methods are the most widely used methods, while among the SEE professionals, parametric models are the dominantly used methods [4]. Although the development of parametric effort models has made the estimation process repeatable, there are still concerns with their overall accuracy [5–8]. Difficulty with accurately sizing software systems is a major cause of this observed inaccuracy.

Unfortunately, it is difficult to get around the sizing problem, because if there is something considered a "law of physics" in the world of SEE; it is that the bigger a system is, the more effort it requires. There are other effort "laws" such as: The more complex a system is, the more effort it requires as well. But in this paper we will focus on system size, especially as most, if not all, of the models used in industry require size as the primary input [9]. The primary sizing metrics used in effort models are logical lines of code (LOC) [9] and function points (FP) [10]. However, there are many other sizing metrics such as physical LOC, number of requirements, number of modules [11], number of web pages [12] and so on. From now on we will refer to such size related metrics generally as *"size features"*. Based on the work done at the NASA Software Engineering Laboratory (SEL), it seems that software systems can be sized in all of these ways and after the task is completed one will find relatively high correlations between the delivered size and actual effort, as long as the data is collected in a consistent and accurate manner.

Part of the sizing problem comes down to the fact that primary sizing metrics have different advantages and disadvantages. For example, LOC can be automated for counting and is good a posteriori, but is difficult to estimate early on. There exist other problems associated with using LOC as the primary size metric as well, which is a major reason why the FP sizing metric was developed: To provide a way to produce a size metric based on early design information; hence, FP would be more accurate a priori. The problem is that FP cannot be automated and is subjective, even though it has been shown that with training the estimate variation can be

reduced [4]. In any case, both of these methods require significant expertise to be applied successfully.

The heart of the problem is that early on we do not know the characteristics of a system well enough. We often really do not know how big a system will be nor how much the requirements change, nor how much code can effectively be reused, especially when there are major new capabilities that need to be developed. One example of relatively successful attempt to get around this problem at the NASA Jet Propulsion Laboratory (JPL) was to develop an integrated effort and sizing model that uses system level characteristics, e.g. the number of instruments, the data rate and what planet you are going to. It took several years to develop this model and it is "hard-wired" to a specific type of system. Unfortunately, this is a very expensive local solution and not a global one.

So for accurate estimates that need to be made very early in the software life-cycle what is very much needed is a simple global method that can work without size features. Unfortunately, standard SEE methods such as a standard ABE method (1NN) as well as classification and regression trees (CART) cannot be this global method. Our results show that standard SEE methods perform poorly with the lack of size features. In this paper we propose an ABE method (*"pop1NN"*) augmented with a popularity-based instance selection mechanism. pop1NN is a simple method that only requires normalization of an array of numbers and Euclidean distance calculation. Our results show that pop1NN (compared to 1NN) compensates for the lack of size features in 11 out of 13 data sets. A simple method like pop1NN running on data sets without size features can attain the same performance as a more complex learner like CART running on data sets with size features.

Practitioners, who are able to measure size features accurately in all phases of the software development life-cycle should do so. On the other hand, if practitioners lack the resources or means to measure size features, we deter the use of standard methods such as 1NN and CART without any pre or post processor. Our recommendation for circumstances where size features cannot be measured accurately is to use pop1NN-like methods, which can compensate for size features through instance selection.

The rest of this paper is organized as follows: In §2 we summarize the motivation behind this research. §3 discusses the related work in SEE and instance selection. The methodology of this research is presented in §4, which is followed by §5 that explains the algorithms and experiments. In §6 we present our results. The threats to the validity of our results are discussed in §7. The discussion of this research is given in §8 and it is followed by the possible future directions in §9. We conclude with §10.

2. MOTIVATION

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 [13]) 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 [14]) 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.

There are various reasons from which the adoption problem 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 [14, 15].

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 [15]: *"Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs."* A similar notion is also adopted by Dijkstra [16]: *"... 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 in this paper 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, yet can attain performance values as good as methods that use size features. We see the implications of this work 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.

3. RELATED WORK

3.1 Software Effort Estimation

SEE can be defined as the process of estimating the total effort necessary to complete a software project [17]. An extensive systematic review conducted by Jorgensen and Shepperd reports that developing new estimation models is the biggest research topic in SEE since 1980s [13]. Hence, there is a big number of SEE models and a taxonomy is necessary to classify such a large corpus. According to Myrtveit et al. taxonomy is an explanation of a concept, which highlights the similarities and differences between that particular concept and the others [18].

There are different taxonomies for SEE methods [18, 19]. Briand et al. report that there is no agreement on the best taxonomy [19]. For example Menzies et al. divide SEE methods into two groups: Model-based and expert-based [20]. According to this taxonomy model-based methods use some algorithm(s) to summarize old data and to make predictions for the new data. Expert-based methods make use of human expertise, which is possibly supported by process guidelines and/or checklists. Myrtveit et al. propose a dataset dependent differentiation between methods [18]. According to that taxonomy the methods are divided into two:

- Sparse-data methods that require few or no historical data: e.g. expert-estimation [21].
- 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 [22]: 1) expert-based estimation, 2) algorithmic models and 3) analogy. Expert based models target the consensus of human experts. Jorgensen et al. define expert-based methods as a human-intensive process of negotiating the estimate of a new project [21]. There are formal methods

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

Table 1: The 494 projects used in this study come from 13 data sets. Indentation in column one denotes a dataset that is a subset of another dataset.

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
nasa93c1	17	12	nasa93 projects from center 1	months
nasa93c2	17	37	nasa93 projects from center 2	months
nasa93c5	17	40	nasa93 projects from center 5	months
desharnais	10	81	Canadian software projects	hours
desharnaisL1	10	46	projects developed with Language1	hours
desharnaisL2	10	25	projects developed with Language2	hours
desharnaisL3	10	10	projects developed with Language3	hours
sdr	22	24	Turkish software projects	months
		Total: 494		

proposed for expert-based estimation like Delphi [23]. However, Shepperd et al. notes in another study that companies mostly follow an informal process for expert-based estimation [24]. Algorithmic models include the adaptation of a formula to local data. Prominent examples to these methods are the COCOMO [2] and FP [25]. Analogy based methods find similar past projects, then adapt the effort values for the current project.

3.2 Instance Selection in SEE

The proposed method in this study is a standard ABE method augmented with a popularity based instance selector. Our results show that a standard ABE method is incapable of dealing with the lack of size features. However, the same ABE method augmented with instance selection can compensate for the lack of size features. We hypothesize that instance selection makes the signal within the data clearer. For example in a prior study Kocaguneli et al. use a variance-based instance selection mechanism in an ABE context [26]. In this variance-based instance selection mechanism instances of a data set are clustered according to their distances and only the clusters with low dependent variable variance are selected. The estimates made from the remaining clusters have a much lower error rate than standard ABE methods.

The fundamental idea behind instance selection is that most of the instances in a data set are uninformative and can be removed. In [27] Chang finds prototypes for nearest neighbor classifiers. Chang's prototype generators explore data sets A,B,C of size 514, 150, 66 instances, respectively. He converts A, B, C into new data sets A', B', C' containing $34, 14, 6$ prototypes, respectively. Note that the new data sets contain only 7%,9% and 9% of the original data.

There are a number of instance selection studies in SEE. Keung et al.'s Analogy-X method also works as an instance selection method in an ABE context [28]. Analogy-X selects the instances of a data set based on the assumed distribution model of that data set, i.e. only the instances that conform to the distribution model are selected. Another example is Li et al.'s study, where they use a genetic algorithm for instance selection to provide estimation accuracy improvements [29]. Turhan et al. use instance selection as a filtering mechanism to enable cross-company data usage [30] in software defect prediction. This study is followed by Kocaguneli et al. where they propose another instance selection filter for SEE cross-company data usage [31].

Table 2: 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**

4. METHODOLOGY

4.1 Datasets

The data used in this study is available at `http://promisedata.org/data` or through the authors (sdr data set). See in Table 1 that a variety of different data sets are used in this research. The standard **COCOMO** data sets (cocomo*, nasa*) contain contractor projects developed in USA and are collected with the COCOMO approach [2]. The **desharnais** data set and its subsets are collected with FP approach from software companies in Canada [32]. **sdr** contains data from projects of various software companies in Turkey and it is collected by SoftLab with COCOMO approach [33].

Using the data sets of Table 1 we would like to define two keywords that will be fundamental to our discussion: *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 (for the experiments of this paper) will be called a *"reduced data set."* The features of the full data sets are given in Table 2. Note in Table 2 that the full data sets are grouped under 3 categories (under the *"Methodology"* column) depending on their collection method: COCOMO [2], COCOMOII [9] and FP [25, 32]. 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 only the additional cost driver features for COCOMOII under the column *"Features"*.

The bold-font features in Table 2 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 3. Note that both in Table 2 and in Table 3, 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 [2, 9]. 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 [32] or Li et al. [34]. 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 [35].

Table 3: 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 2).

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

4.2 Error Measures

Error measures are used to assess the success of a prediction. The absolute residual (AR) is defined to be the difference between the predicted and the actual:

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

(where $x_i, \hat{x_i}$ are the actual and predicted value for test instance i). MAR is the mean of all the AR values.

The Magnitude of Relative Error measure (MRE) is one of the de facto error measures [1, 36]. MRE measures the ratio between the effort estimation error and the actual effort:

$$MRE_i = \frac{\mid x_i - \hat{x_i} \mid}{x_i} = \frac{AR_i}{x_i} \tag{2}$$

A related measure is the **M**agnitude of **E**rror **R**elative to the estimate (MER) [36]):

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

The overall average error of MRE can be derived as the Mean or Median of MRE, i.e. MMRE and MdMRE, respectively:

$$MMRE = mean(allMRE_i) \tag{4}$$

$$MdMRE = median(allMRE_i) \tag{5}$$

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

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

For example, PRED(25)=50% implies that half of the estimates are failing within 25% of the actual values [1].

There are many other error measures including Mean Balanced Relative Error (MBRE) and the Mean Inverted Balanced Relative Error (MIBRE) studied by Foss et al. [36]:

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

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

Mere use of error measures without the support of appropriate statistical tests can be misleading [36]. A detailed discussion regarding problems in SEE studies due to the lack of statistical tests in method comparisons is provided by Kitchenham et al [37]. In this research we employ Mann-Whitney Rank-Sum test (95% confidence). We use Mann-Whitney instead of Student's t-test, as as it compares the sums of ranks, unlike Student's t-test, which may introduce spurious findings as a result of outliers in the given datasets. Also, non-parametric tests are useful when Gaussian assumption about the underlying distributions is not clear [38].

We use Mann-Whitney to generate $win, tie, loss$ statistics. The procedure used for that purpose is defined in Figure 1: Firstly two distributions i, j (e.g. arrays of MRE's or AR's) are compared to see if they are statistically different (Mann-Whitney rank-sum test, 95% confidence); if not, then the related tie values (tie_i and tie_j) are incremented. If the distributions are different, we update win_i, win_j and $loss_i, loss_j$ after error measure comparison.

When comparing the performance of a method ($method_i$) run on reduced data sets to a method $method_j$ run on full data sets, we will specifically focus on $loss$ values. Because note that to make the case for $method_i$, we do not need to show that its performance is *"better"* than $method_j$. We can recommend $method_i$ over $method_j$ as long as its performance is no worse than $method_j$, i.e. as long as it does not *"lose"* against $method_j$.

if Mann-Whitney(Err_i, Err_j, 95) says there is no statistical difference. **then**
  tie_i = tie_i + 1;
  tie_j = tie_j + 1;
else
  if better(Err_i, Err_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 1: Comparing error measures of method (*i,j*) (Err_i,Err_j). The "better" predicate changes according to error measures; e.g. for MRE, "better" means lower values, whereas for PRED(25), "better" means higher values.

5. EXPERIMENTAL CONDITIONS

5.1 The Algorithms Adopted

In this study two algorithms are used: 1) a baseline analogy-based estimation method called ABE0 [39] and 2) a decision tree learner called **C**lassification **A**nd **R**egression **T**rees (CART) [40]. Our justification for the selected learners is based on prior work in SEE. Several papers conclude that CART and nearest neighbor methods are useful comparison algorithms for SEE. For example, Walkerden & Jeffrey [41] endorse CART as a state-of-the-art SEE method. Dejaeger et al. also claim that, in terms of assessing new SEE methods, methods like CART may prove to be more adequate. The work of Dejaeger et al. has found that learners more elaborate than CART fall short of offering any significant value-added.

Our own results support the conclusions of Walkerden, Jeffrey, Dejaeger et al. Our 2011 study reports an extensive comparison of 90 SEE methods (generated using all combinations of 10 preprocessors and 9 learners) [42]. In this study the most successful SEE methods turn out to be ABE0 with 1 nearest-neighbor (which will be referred as 1NN from now on) and CART.

ABE methods generate an estimate for a future project by retrieving similar instances from a database of past projects. Then the effort values of the retrieved past projects are adapted into an estimate. There are various design options associated with ABE methods. Reading from the ABE methods used in Kadoda & Shepperd [43], Mendes et al. [44], and Li et al. [29], here we define ABE0, which is a *baseline* ABE method that works as follows:

- 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 9, where w_i corresponds to feature weights applied on independent features fA_i and fB_i of instances A and B, respectively. t is the total number of independent features. ABE0 framework does not favor any features over the others, i.e. $w_i = 1$. For adaptation ABE0 takes the median of retrieved *k* projects. The ABE0 adopted in this research uses a 1 nearest neighbor (i.e. *k*=1), hence the name 1NN.

$$Distance = \sqrt{\sum_{i=1}^{t} w_i (fA_i - fB_i)^2} \quad (9)$$

Iterative dichotomizers like CART find the feature that most divides the data such that the variance of each division is minimized [40]. The algorithm then recurses into each division. Once the tree is generated, the cost data of the instances in the leaf nodes are averaged to generate an estimate for the test case. For more details on CART refer to Breiman et al. [40]

5.2 Proposed Method: pop1NN

In this study we propose 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:
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 $Train'$.

For the error measure in *Step 5*, we used "MRE", which is only one of the many possible error measures. As shown in §6, even though we guide the search using only MRE, the resulting estimations score very well across a wide range of error measures. 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 (n, Δ) values can be a promising future work.

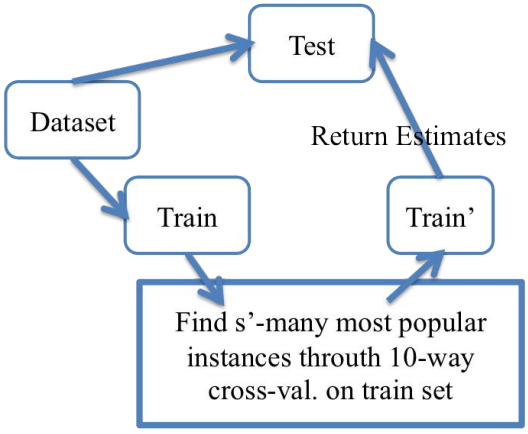

Figure 2: A simple illustration of the pop1NN method. Note that the test and train sets are generated through a 10-way cross-validation as well.

5.3 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 see in §6 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.

6. RESULTS

6.1 Results Without Instance Selection

Table 4 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 4 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 4 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 4 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 4 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 4 for all the methods in both stages of the experimentation is cumbersome and would redundantly take too much space in this paper. Hence, from now on we will use summary tables as given in Table 5, which shows only the total number of losses. See that the *"CART"* column of Table 5 is just the last column of Table 4 . Aside from the CART results, Table 5 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 experimented in this paper. 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.

6.2 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 6. Table 6 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 6 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 [45]. 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 6 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 6, 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 val-

Table 4: 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 5: 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

Table 6: 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

ues as good as CART for most of the data sets; 2) the performance of pop1NN comes from data sets without any size features.

7. THREATS TO VALIDITY

Internal validity questions to which extent the cause-effect relationship between dependent and independent variables holds [35]. 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 ideal case of new and unseen data, sampling methods are employed. In this research we employed 10-way cross validation.

External validity questions the generalization of the results, i.e. it asks how widely the results can be generalized [46]. For the purpose of wider external validity, we have used a wide range of data sets from the PROMISE data repository. Table 4 of [47] lists the total number of projects used by a sample of other SEE studies. The median value of that sample is 186; i.e. a fraction of the 494 projects used in this research. In terms of external validity, this paper has higher validity than a standard effort estimation study.

Construct validity (i.e. face validity) questions whether or not we are measuring what we intend to measure [48]. An insightful discussion in the area can be found in [49], where Kitchenham et al. state that different error measures evaluate different aspects of the prediction accuracy. Hence, to maximize construct validity, we use 7 different error measures. Another point made by Kitchenham et al. in another study [37] is that sole usage of error measures is wrong and they need to be supported with statistical checks. To address that validity issue we use the method of Figure 1, where we make use of Mann-Whitney at a significance level of 95%.

A further validity issue arises due to characteristics of the SEE data sets. For example, the effects of correlation between non-size features to size should be investigated further for these data sets. For some data sets, the non-size features may already contain enough information for a learner to select the most similar training examples; whereas, for some other data sets -due to lack of correlation between size and non-size features- the presence of size features may turn out to be a necessity. Investigation of the correlation between non-size and size features can be a good future direction to this study and may reveal confounding factors. Also, the issue of "project size comparison" between training instances selected by pop1NN and test instances may show why the lack of size features can be handled for certain test cases, whereas a minority of the test instances suffer from it.

8. 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 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.

9. FUTURE WORK

An interesting future direction would be to use pop1NN as a feature selector. pop1NN removes the instances of a data set based on their Euclidean distances in a space defined by the independent features. Such a distance based removal mechanism is not restricted to instances and can also serve as a feature selector. Lipowezky [50] notes that feature and case selection are similar tasks, which both remove cells in the hypercube of all instances times all features. According to the viewpoint of Lipowezky, it should be possible to convert a case selection mechanism into a feature selector.

Another future direction we consider to follow is to use pop1NN as a guidance mechanism for experts in data collection studies. Note that pop1NN does not use dependent variable information. Therefore, pop1NN can identify the popular instances of a data set, for which the experts should collect the dependent variable information. In scenarios, where it is cheap to collect independent variable information but expensive to collect dependent variable information, pop1NN like methods may serve as guidance mechanisms.

Lastly, it is possible to use the popularity based pre-processing mechanism introduced in this paper for other learners. Here, we used the popularity counts as a pre-processor for 1NN, however it may as well be used for other learners.

10. CONCLUSION

Size features are fundamental to many estimation methods such as COCOMO, COCOMOII, FP and so on. In this research 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 [42]. 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 (in this research it is a popularity based pre-processor), 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.

11. REFERENCES

[1] M. Shepperd and C. Schofield, “Estimating software project effort using analogies,” *IEEE Trans. Softw. Eng.*, vol. 23, no. 11, pp. 736–743, 1997.

[2] B. W. Boehm, *Software Engineering Economics*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[3] K. Lum, J. Powell, and J. Hihn, “Validation of spacecraft cost estimation models for flight and ground systems,” in *ISPA’02: Conference Proceedings, Software Modeling Track*, 2002.

[4] J. Hihn and H. Habib-agahi, "Cost estimation of software intensive projects: a survey of current practices," in *Proceedings of the 13th international conference on Software engineering*, ser. ICSE '91. Los Alamitos, CA, USA: IEEE Computer Society Press, 1991, pp. 276–287. [Online]. Available: http://dl.acm.org/citation.cfm?id=256664.256780

[5] T. Menzies, D. Port, Z. Chen, J. Hihn, and S. Stukes, "Validation methods for calibrating software effort models," in *ICSE '05: Proceedings of the 27th international conference on Software Engineering*. New York, NY, USA: ACM, 2005, pp. 587–595.

[6] K. Molokken and M. Jorgensen, "A review of surveys on software effort estimation," in *ISESE '03: Proceedings of the 2003 International Symposium on Empirical Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 223–. [Online]. Available: http://dl.acm.org/citation.cfm?id=942801.943636

[7] D. Ferens, "The conundrum of software estimation models," in *NAECON'98: Proceedings of the IEEE 1998 National Aerospace and Electronics Conference*, 1998, pp. 320 –328.

[8] C. Kemerer, "An empirical validation of software cost estimation models," *Communications of the ACM*, vol. 30, no. 5, pp. 416–429, May 1987.

[9] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, *Software Cost Estimation with Cocomo II*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[10] A. Albrecht and J. Gaffney, "Software function, source lines of code and development effort prediction: A software science validation," *IEEE Trans. Softw. Eng.*, vol. 9, pp. 639–648, 1983.

[11] I. Herraiz, G. Robles, J. Gonzalez-Barahona, A. Capiluppi, and J. Ramil, "Comparison between slocs and number of files as size metrics for software evolution analysis," in *CSMR06: Proceedings of the 10th European Conference on Software Maintenance and Reengineering*, 2006, pp. 8 –213.

[12] E. Mendes, *Cost Estimation Techniques for Web Projects*. Hershey, PA, USA: IGI Publishing, 2007.

[13] 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.

[14] E. Kocaguneli, A. Misirli, B. Caglayan, and A. Bener, "Experiences on developer participation and effort estimation," in *SEAA'11: 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2011, pp. 419 –422.

[15] K. Gollapudi, "Function points or lines of code? - an insight," in *Global Microsoft Business Unit, Wipro Technologies*, 2004.

[16] E. W. Dijkstra, "On the cruelty of really teaching computing science," 1988. [Online]. Available: http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1036.PDF

[17] 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

[18] 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.

[19] L. Briand and I. Wieczorek, *Resource Modeling in Software Engineering*, 2nd ed., ser. Encyclopedia of Software Engineering. Wiley, 2002.

[20] 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.

[21] M. Jorgensen, "A review of studies on expert estimation of software development effort," *Journal of Systems and Software*, vol. 70, pp. 37–60, 2004.

[22] 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.

[23] G. Rowe and G. Wright, "The delphi technique as a forecasting tool: issues and analysis," *International Journal of Forecasting*, vol. 15, 1999.

[24] M. Shepperd and M. Cartwright, "Predicting with sparse data," *IEEE Trans. Softw. Eng.*, vol. 27, no. 11, pp. 987–998, 2001.

[25] K. Atkinson and M. Shepperd, "The use of function points to find cost analogies," *European Software Cost Modelling Meeting . Ivrea, Italy*, 1994.

[26] 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, pp. 425–438, 2011.

[27] C. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Trans. on Computers*, vol. C23, pp. 1179–1185, 1974.

[28] J. Keung, "Empirical evaluation of analogy-x for software cost estimation," in *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. New York, NY, USA: ACM, 2008, pp. 294–296.

[29] 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.

[30] 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, pp. 540–578, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10664-008-9103-7

[31] 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, pp. 255 –264.

[32] 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.

[33] A. Bakir, B. Turhan, and A. Bener, "A new perspective on data homogeneity in software cost estimation: A study in the embedded systems domain," *Software Quality Journal*, 2009. [Online]. Available: http://dx.doi.org/10.1007/s11219-009-9081-z

[34] 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

[35] E. Alpaydin, *Introduction to Machine Learning*. MIT Press, 2004.

[36] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion mmre," *IEEE Trans. Sofw. Eng.*, vol. 29, no. 11, pp. 985–995, Nov. 2003. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1245300

[37] 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.

[38] J. P. C. Kleijnen, "Sensitivity Analysis and Related Analyses: A Survey of Statistical Techniques," *Journal Statistical Computation and Simulation*, vol. 57, no. 1–4, pp. 111–142, 1997.

[39] E. Kocaguneli, G. Gay, T. Menzies, Y. Yang, and J. Keung, "When to use data from other projects for effort estimation," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2010, pp. 321–324. [Online]. Available: http://portal.acm.org/citation.cfm?id=1859061

[40] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[41] 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.

[42] E. Kocaguneli, T. Menzies, and J. Keung, "On the value of ensemble effort estimation," *IEEE Trans. Softw. Eng.*, vol. Preprints, 2011.

[43] 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.

[44] 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.

[45] 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.

[46] D. Milic and C. Wohlin, "Distribution patterns of effort estimations," in *Euromicro*, 2004.

[47] 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.

[48] C. Robson, "Real world research: a resource for social scientists and practitioner-researchers," *Blackwell Publisher Ltd*, 2002.

[49] B. A. Kitchenham, L. M. Pickard, S. G. Macdonell, and M. J. Shepperd, "What accuracy statistics really measure," *IEEE Proceedings-Software*, vol. 148, pp. 81 – 85, 2001.

[50] U. Lipowezky, "Selection of the optimal prototype subset for 1-NN classification," *Pattern Recognition Letters*, vol. 19, no. 10, pp. 907–918, 1998. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167865598000750