TEAK: Learning Better Case Selection Strategies for Analogy Based Software Cost Estimation

Ekrem Kocaguneli, *Student Member, IEEE,* Tim Menzies, *Member, IEEE,* Ayse Bener, *Member, IEEE,* and Jacky W. Keung, *Member, IEEE*

Abstract—*Background:* Software effort estimation has been one of the major challenges in software engineering. Previous models are evaluated using measures such as MRE and pred(r), which assess performance on the basis of prediction accuracy.
Objective: Instead of focusing on algorithmic complexity, there is a need for 1)understanding the data and 2) providing accurate estimations.
Method: We adapt greedy agglomerative clustering (GAC) algorithm to software effort estimation and use it as an analogy based estimator to build our model: Tree Estimation and Assessment Knowledge(TEAK).
Result: TEAK provides an analogy number for each test project and get lower MRE values than any other *k*-based method.
Limitation: There are multiple problems with case based reasoning (CBR) methods such as feature subset selection, and number analogies to use (*k* value)[1]. As our intention is to focus on finding the suitable *k* value, we do not address other CBR related problems.
Conclusion: With TEAK it is possible to better understand the data, get rid of irrelevant analogies and use different number of analogies for each test instance. This approach has outperformed all other *k*-based CBR methods upto 69% in terms of pred(25) values.

Index Terms—Software Cost Estimation, Greedy Agglomerative Clustering, *k*-NN

1 Introduction

One of the key challenges in software industry is the accurate estimation of the development effort, which is particularly important for risk evaluation, resource scheduling as well as progress monitoring[1]. Inaccuracies in estimations leads to problematic results, for instance overestimation causes waste of resources, whereas underestimation results in approval of projects that will exceed their planned budgets[2].

Although a significant number of methodologies have been proposed for effort estimation over the years, they have suffered from common problems such as very large performance deviations [3] as well as being highly dataset dependent. Comparative studies regarding best practices have therefore shown contradictory results[4].

In the recent years a significant research effort was put into utilizing various machine learning (ML) algorithms as a complementary or as a replacement to previous methods[1][5][6][7]. However, ML methods have an extremely large space of configuration possibilities[1]. When we consider configuration possibilities of ML methods induced on different datasets each having a characteristic of its own, it is not a surprise to see contradictory results[4][1].

The predictive performance of any method is dataset dependent. Previous research has reported case based reasoning(CBR) or estimation by analogy[1] being able to produce more successful results in comparison to traditional regression based methods[7][8]. Furthermore CBR has been favoured over other methods when the dataset contains discontinuities[1]. However, it was also remarked that CBR techniques are subject to a variety of decisions that have a strong impact on its predictive performance. Such decisions include selection of features and/or instances, deciding on the number of analogies to be used and the adaptation strategy[1]. In this paper, we focus on deciding the number of analogies to be used (i.e. *k* nearest neighbours or projects[1]). Our claim is that we can avoid sticking to a fixed best performing number of analogies that changes from dataset to dataset. We use an alternative method to tune CBR techniques by proposing greedy agglomerative clustering algorithm (GAC).

GAC starts with the single project instances within the train dataset and treats them as the leaves of a GAC tree. Then it iteratively combines closest two projects to form a parent node, the same procedure applies for the parent nodes to build their parents and so on. At the end we end up with a binary tree build by GAC, whose leaves are

- *Ekrem Kocaguneli is with the Department of Computer Engineering, Bogazici University*
  E-mail: ekrem.kocaguneli@boun.edu.tr
- *Ayse Bener is with the Department of Computer Engineering, Bogazici University.*
  E-mail: bener@boun.edu.tr
- *Tim Menzies is with the Lane Department of Computer Science and Electrical Engineering, West Virginia University.*
  E-mail: tim@menzies.us
- *Jacky W. Keung is with the School of Computer Science and Engineering, University of New South Wales.*
  E-mail: jacky.keung@nicta.com.au

This research is funded in part by Tubitak EEEAG108E014.

the actual project instances and whose nodes are clusters of those single projects.

In our study, we form two GAC trees from the project data within our train set to build our model tree estimation and assessment knowledge (TEAK). The first GAC tree (GAC1) is built on all the available train data and is used to decide on the node that has the lowest performance-variance, thereby pruning irrelevant train instances that would otherwise introduce higher variance and lower prediction accuracies on the results. The train instances which were clustered in the lowest performance-variance node are then used for building up the second GAC tree (GAC2). At the end our estimation becomes the median of the projects that are clustered in the lowest performance-node of GAC2.

Our aim by utilizing two GAC trees is to introduce a CBR calibrating method that makes its estimations via 1) building itself by discovering the characteristics of a particular dataset on its own and 2) pruning irrelevant instances on the basis of performance-variance. Our rigorous experiments that were conducted on 10 highly dissimilar software effort datasets have shown that GAC2 estimations are about 69% less error prone than any fixed-*k* analogy approach (*k* is the number of analogies) on the basis of magnitude of relative error (MRE) subject to 95% Wilcoxon signed rank test.

The rest of the article is organized as follows: Section 2 provides related work. Section 3 defines the main problem discussed in this paper and the methodology we propose to address this problem. Section 4 presents the results we obtained. Lastly Section 5 summarizes our conclusions and future work.

2 Background

Software effort estimation has been extensively studied in literature since 70's. We can group effort estimation methods under two main categories: 1) Expert based methods and 2) model based methods, where the former proposes making use of the experiences of human experts and the latter favouring algorithmic approaches. The first model based methods were parametric models[9][10][11]. Although they generate successful results in certain datasets, parametric approaches suffered from local tuning problems when they were to be applied in another setting, i.e. they needed to be tuned to local data for high accuracy values[12]. On the other hand, local tuning requires collection of local data and the whole measurement and collection process is a challenge on its own[12][3].

To address the local tuning problems of regression based methods, machine learning (ML) algorithms have been proposed as they do not require local tuning. Basically ML based models learn from the past projects' effort data to make predictions for the future projects[13][7][14][15].

However, industry's practices diverge from promising highly complicated formal methods and a software project manager's mental process of effort estimation is closer to CBR methods[16]. Indeed from our hands on experiences in software effort estimation model building in industry settings, we end up with similar conclusions as well. Furthermore, software effort datasets are characteristically noisy datasets and CBR methods are more capable of handling noisy datasets than regression based models[16].

CBR can be defined as predicting the effort of a project by making use of similar past projects (analogies). To find the analogies, a number of methods can be employed: Nearest neighbour algorithms like *k*-NN, expert guidance or goal based preference[7]. To predict the number of analogies (*k*) while making an estimation is a challenge with CBR methods, since the number *k* plays a very critical role in the success of these methods[1]. In literature there are a number of studies proposing different *k* values in different contexts. Babu et. al. addresses the issue of selecting prototypes, which they define as a process of finding representative patterns from data[17]. In their study Babu et. al. uses static selection rules to determine the best neighbourhood around a test instance and such rules show uniformity for all test set. On the contrary, we propose a selection scheme that uses dynamic rules, which are tuned per each single test instance. A further approach proposed by Huang et. al. for prototype optimization for *k*-NN uses a neural-net-based method.[18]. Method of Huang et. al. is capable of updating more than one prototype in case of a misclassification and constructs not only the prototypes but also feature weights during the selection and optimization process. Therefore, from a theoretical point of view there is much space for adding in more machinery into standard estimation methods. However, there is hardly any strong evidence that this is useful in software effort estimation. What is more, studies that involve much machinery induced on limited and not publicly accessible datasets may reduce the reproducibility of experiments as well as their external validity. In our study we are offering a prototype selection tool that uses single data structure called GAC tree twice (for GAC1 and GAC2).

Some studies favour using a pre-determined number of analogies in software effort estimation studies[19][20][21]. Lipowezky et. al. proposes a policy that looks for only one prototype, which can be regarded as extreme when dealing with datasets as small as those in software effort estimation[19]. Furthermore, we would like to base our estimations on some sample set of past data, not only on one record, since only one record may be misleading in small and heterogeneous datasets. Kirsopp et. al. on the other hand proposes making predictions from the 2 nearest cases as it was found as the optimum value for the datasets of their study[20]. In a further study Kirsopp et. al. have increased their accuracy values with case and feature subset selection strategies[21]. Leaving the feature selection to future work, with dynamic selection of instances we have attained higher accuracy values than those reported for static rules or methods. Another study focusing on instance selection as well as feature weighting in context of CBR is conducted by Li et. al. [22]. In their study Li et. al. perform rigorous trials on actual and artificial datasets and they observe effect of various k values. However, we believe that reflection on dataset before applying to different algorithms under multiple settings is of more significance. In this study we propose making use of greedy agglomerative clustering (GAC) to address this challenge.

GAC was previously utilized in various areas: Data mining[23], database systems[24] and bioinformatics[25]. The agglomerative (bottom-up) clustering method we use in this research groups similar individual projects into bigger clusters until a root is formed[26]. To find the similarity between projects, we use Euclidean distance. In our proposed model, GAC is used in software effort estimation domain to remove noisy data points and to automatically come up with the number of analogies to be used.

3 Methodology

In this paper we are proposing a new approach to CBR calibration methods: Greedy Agglomerative Clustering. To the best of our knowledge, it has not been used previously in software effort estimation domain. While proposing our method for CBR calibration, we would like to address the problem of number analogies to use (*k* value) among the previously reported issues related to CBR methods. In a study conducted by Kadoda et. al[1], the impact of the number of analogies selected while making predictions were investigated on software effort data and it was reported that the underlying distribution of dataset had a decisive role on that number. Furthermore, they also suggested that datasets come with their inherent complex properties and it requires a lot of time and effort to adapt a CBR system to a particular dataset[1]. In our study we would like to answer the following research questions.

1. How can we better understand the dataset and the underlying characteristics of the dataset?
2. How can we ease the procedure of selecting the number of analogies to be used and improve the performance of current CBR methods?

We propose a model called TEAK on the basis of performance-variance. We compare our model with other methods in terms of magnitude of relative error (MRE), prediction at level r (pred(r)) and *"win-tie-loss"* values. Each of these performance measures will be defined in detail in Section 3.4.

3.1 Data

Dataset characteristics are very important to evaluate the performance of a model. Since many previous models are criticized for having been tried on single or a very limited number of datasets[4][1], we conducted our experiments on 10 different datasets coming from 3 different sources.

The first source we have used is PROMISE data repository[27]. PROMISE data repository is an on-line publicly available data repository and it consists of datasets donated by various researchers around the world. The datasets we have elicited via PROMISE data repository are: Cocomo81[9], Nasa93[28], Desharnais[29] and Albrecht[30].

Another source of data we have made use of is the Bogazici University Software Engineering Research Laboratory repository (SoftLab)[6]. The dataset we have taken from SoftLab is SDR and it contains projects of various software companies from Turkey.

The last source we have used is the International Software Benchmarking Standards Group (ISBSG). ISBSG is a non-profit organization and it maintains ISBSG dataset, which is a project management dataset consisting of contributions from various companies and organizations[31].

In our study we also wanted to see the performance of our model TEAK on homogeneous datasets in terms of a particular property. Therefore we only selected homogeneous datasets that are as big as the smallest heterogeneous dataset in terms of instance number.

Cocomo81 dataset enables the researchers to classify projects in terms of three different development modes: Organic, semi detached and embedded [9]. Therefore we used development mode as our breakdown criteria in Cocomo81 and took two homogeneous subsets of Cocomo81: Cocomo81o and Cocomo81e. Cocomo81o includes organic projects and Cocomo81e includes embedded projects.

For Nasa93 dataset our breakdown criteria was the development center of projects. Projects in Nasa93 are developed in one of the six different development centers. We took two subsets of Nasa93 dataset on the basis of their development centres: Nasa93c2 and Nasa93c5. Nasa93c2 is composed of projects that were developed in center 2 whereas Nasa93c5 contains projects that were developed in center 5.

Projects in ISBSG dataset can be grouped according to their business domains. In previous studies, breakdown of ISBSG according to business domain has also been used[32]. Among different business domains we selected banking due to:

1. Banking domain includes many projects whose data quality is reported to be high (ISBSG contains projects with missing attribute values).
2. ISBG Banking domain is the dataset we have analyzed and worked for a long time due to our hands on experience in building effort estimation models in banking industry.

We will represent the banking domain subset of ISBSG as ISBSG-Banking.

We provide further details regarding 10 datasets that we used in our research in two categories: 1) General properties and 2) statistical properties. General properties include name of the dataset which we use throughout the paper, number of features and the number of instances within the dataset, whereas statistical properties include statistical facts concerning the independent variable (effort) such as mean, median, minimum, maximum and skewness. General properties of datasets can be found in Table 1 and statistical properties are reported in Table 2.

TABLE 1: General Properties of Utilized Datasets

Dataset	Features	Projects	Content
Cocomo81	17	63	NASA projects
Cocomo81e	17	28	Cocomo81 embedded projects
Cocomo81o	17	24	Cocomo81 organic projects
Nasa93	17	93	NASA projects
Nasa93c2	17	37	Nasa93 projects from center 2
Nasa93c5	17	40	Nasa93 projects from center 5
Desharnais	12	81	Canadian software projects
SDR	22	24	Turkish software projects
Albrecht	7	24	Projects from IBM
ISBSG-Banking	14	29	Banking projects of ISBSG

TABLE 2: Statistical Properties of Utilized Datasets

Dataset	Mean	Median	Min	Max	Skewness
Cocomo81	683.52	98.00	5.90	11400	4.36
Cocomo81e	1152.80	354.00	9.00	11400	3.37
Cocomo81o	59.87	46.00	6.00	240.00	1.68
Nasa93	624.41	252.00	8.40	8211.00	4.18
Nasa93c2	222.91	82.00	8.40	1350.00	2.37
Nasa93c5	1011.10	571.40	72.00	8211.00	3.46
Desharnais	5046.30	3647	546	23940	1.96
SDR	32.04	12.00	2	342	3.93
Albrecht	21.87	11.45	0.5	105.20	2.15
ISBSG-Banking	5357.00	2355.00	662	36046	2.62

As we can see from Table 2, all the datasets have positive skewness and the skewness values range from 1.96 to 4.36, which indicates that the datasets are extremely heterogeneous with as much as 40-fold variation. We can conclude that the datasets diverge both in terms of their statistical characteristics as well as their domains and sizes. Therefore we make sure that we test the proposed model adequately.

3.2 TEAK Model

Agglomerative clustering is a commonly used clustering method in various fields including data mining[23], database systems[24] and bioinformatics[25]. The popularity of making use of agglomerative clustering is its ability to use arbitrary dissimilarity or distances functions[26], which also makes it an appealing choice of clustering method for software effort data as software effort datasets also exhibit very dissimilar characteristics.

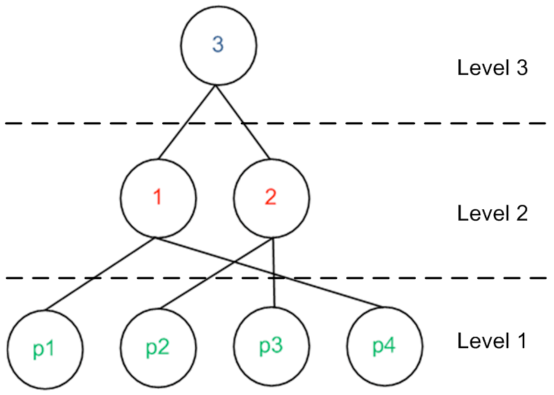


Fig. 1: A Simple GAC Tree Example

Greedy agglomerative clustering (GAC) is a greedy algorithm that starts with a set of instances and builds up a binary clustering tree with those instances according to a distance function[26]. In our research we use Euclidean distance function while building up our GAC tree. The formula of Euclidean distance function is given in Equation 1, where x and y correspond to two different projects, whereas x_i and y_i correspond to their i^{th} features.

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

At the beginning of GAC, we can think of each individual project as a cluster of 1 instance. GAC then calculates the Euclidean distance between every two instance and greedily clusters closest two projects to form a cluster in one higher level of the tree. Then GAC continues to follow the same procedure at a higher level and so on until all the instances are clustered at a central cluster. Since the final structure is a binary tree as can be seen from Figure 1, we will use the terms cluster and node/root interchangibly from now on. Figure 1 is a very simple GAC tree that can be formed on a simple dataset of 4 projects. Let us assume that p1, p2, p3 and p4 are four initial projects of a hypothetical software effort dataset and they form Level 1 of the GAC tree. In that dataset p1-p4 and p2-p3 form the closes pairs of projects. Therefore, they are clustered to form the nodes at a higher level, Level 2. Finally node 1 and node 2 are clustered to form the node at Level 3 and since there are no more nodes to cluster, we are done building up the GAC tree.

In this research we are building up two GAC trees that are very similar to GAC tree of Figure 1. The first GAC (GAC1) tree we build uses n-1 instances of a software effort dataset of size n and uses 1 instance for test. Then all the subtrees of GAC1 are traversed and their variances are stored. So as to remove the bias coming from numeric differences, stored subtree variances are normalized. Then TEAK probabilistically selects nodes according to their normalized variance values. The actual

```
*currentLevelNodes* = *Training Instances*
*levelIndex* = 1
*tree.level(levelIndex).nodes* = *currentLevelNodes*
{As long as we did not get to root, continue forming GAC}
**while** size(*currentLevelNodes*) $>$ 1 **do**
    *nextLevelNodes* = null
    *levelIndex* = *levelIndex* + 1
    **repeat**
        [*closest1*,*closest2*] = findClosest(*currentLevelNodes*)
        *nodeToAdd* = combineAsNewNode(*closest1*,*closest2*)
        *tree.level(levelIndex).nodes*.add(*nodeToAdd*)
        deleteFromCurrentLevelNodes(*closest1*,*closest2*)
    **until** size(*currentLevelNodes*) == 0
    *currentLevelNodes* = *tree.level(levelIndex).nodes*
**end while**
```

Fig. 2: GAC Pseudocode

```
**if** $normalizedVarianceNode_i \leq rand()^{bias}$ **then**
    $GAC2 \leftarrow GAC2 + Node_i$
**end if**
```

Fig. 3: Probabilistic Selection Pseudocode

project instances that were used to create the selected subtrees are used to form the second GAC tree (GAC2). Furthermore, TEAK enables its user to fine-tune GAC2 tree with the help of a variables called *bias* that represents the user's expert bias. The probabilistic selection of nodes from GAC1 as well as how *bias* works can be seen from pseudocode given in Figure 3.

We call our model, which is a special form of two cascaded GAC trees as **T**ree **E**stimation and **A**ssessment **K**nowledge(TEAK).

Finally once GAC1 and GAC2 are built, the estimation process starts. For estimation the test instance moves along the tree starting from the root until it finds the so called *lowest-performance-variance node*. The decision whether a test instance will move from one node to another node in the tree depends on comparison of variances of the current node and the weighted variance of its subtrees. Given three subtrees containing $N = N1 + N2 + N3$ leaf nodes with variance values $V1, V2, V3$ then weighted variance beneath a node is calculated using Equation 2.

$$WeightedVariance = \frac{V1 * N1}{N} + \frac{V2 * N2}{N} + \frac{V3 * N3}{N} \quad (2)$$

If weighted variance of subtrees beneath a node is smaller than that of the node, then the test instance chooses to move to either left or right child that has the the lowest variance. On the other hand, if the weighted varince of subtrees is larger than that of their root node, test instance stays at that node. Finally the median of the train instances that form the node at which test instance stops becomes the estimated effort value for that test instance, i.e. we choose to use *k*-many analogies for estimation where *k* is the number of train instances that are in the selected node. The node at which the test instance stops is called the lowest-performance-variance node.

```
Dataset = *Cocomo81, Cocomo81e, Cocomo81o, Nasa93, Nasa93c2, Nasa93c5, Desharnais, Albrecht, SDR, ISBSG-Banking*
**for** $i = 1$ to 20 **do**
    $Dataset = randomize(Dataset)$
    **for all** $T$ in Dataset **do**
        $testSet$ = $T$
        $trainSet$ = $Dataset$ minus $T$
        $GAC1$ = build GAC tree on $trainSet$
        $selectedNode$ = start from root and select lowest variance node from $GAC1$
        $GAC2$ = build GAC tree on unique instances of $selectedNode$
        $finalNode$ = start from root and select lowest variance node from $GAC2$
        predicted effort for $T$ = $median(finalNode)$
    **end for**
**end for**
```

Fig. 4: TEAK Pseudocode

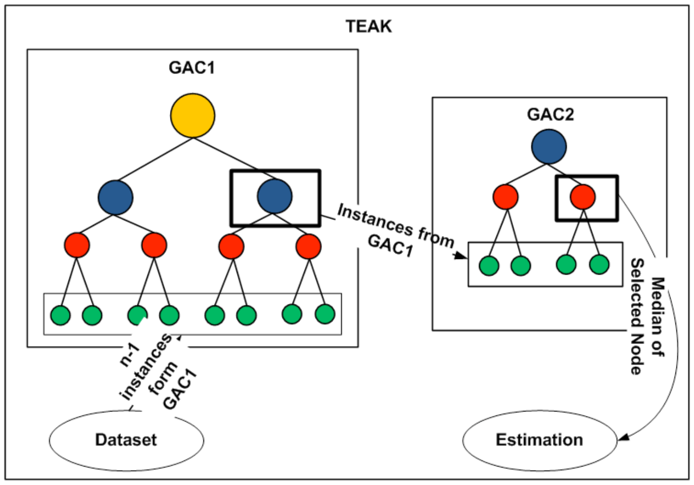


Fig. 5: Execution of TEAK

The aforementioned estimation procedure is performed for each individual instance in an effort dataset separately. In other words, each instance within the dataset becomes a test instance for once and the remaining instances are used as its train instances to build the two GAC trees in TEAK. To better understand the TEAK algorithm, we provide the pseudocode in Figure 4 and we also provide a schema of the algorithm in Figure 5.

Our proposed model, TEAK, may use more than two GAC trees, since the dataset itself usually defines the number of GAC trees to be used. As long as we have enough instances to build a third or fourth GAC tree, there is no number limit for the GAC trees. However, since the size of our ten datasets in this study range from 24 instances to 93 instances, the instances left for building up GAC2 can be as low as 2. Therefore, due to characteristics of our effort datasets, TEAK uses 2 GAC trees. Tricky point here is that unlike *k*-based CBR methods, TEAK does not need any expert interference to discover dataset characteristics so as to decide on the number of trees to be built or the number of analogies to be used in estimation.

Regular *k*-based CBR methods start with single analogies and increase this number depending on the overall performance of the whole dataset. Finally it uses the *k* value that yields the overall best performance. However, a fixed *k* value yielding *overall* best performance, does not necessarily provide the best performance for individual projects. We provide further details about the difference between individual and overall best performing *k* values in Section 4. Unlike regular *k*-based CBR methods, TEAK starts with all the instances in the train set and gradually prunes irrelevant instances on the basis of performance-variance. Furthermore, the pruning of irrelevancies is not performed for overall dataset but for each individual test instance, thereby considering the individual performances of each instance.

In that respect TEAK may resemble Ada Boost algorithm[33]. Ada Boost is basically composed of cascaded learners. At each round of Ada Boost, training set for a learner is selected probabilistically from the examples on which the previous learner performed poorly. Therefore, the selection criteria becomes the score of performances. The decision making then is performed by combination of learners via voting. The first difference of TEAK from standard Ada Boost is that it defines the number of learners (GAC trees) it will use by discovering the properties of the dataset. This feature makes it a better choice for software effort domain as datasets have varying characteristics. Secondly, while still probabilistically selecting train instances for GAC2, TEAK uses performance-variance. Finally, TEAK does not use voting of all learners. It uses the prediction of the the last learner (GAC2) as the prediction of the model.

3.3 Experimental Design

We used ten different datasets from three different sources in our experiments to ensure that we address the criticism regarding previous studies in terms of trying models only on a limited number of datasets from limited sources and/domains[34][1][4]. For each dataset we follow the same testing strategy, we use leave-one-out method[33] to select our test instance and use the remaining instances as our training instances. We perform twenty runs on every dataset. In each run we select a random instance from the dataset, run TEAK on training instances and store our estimation, then we select another random instance and re-run TEAK. This estimation procedure is performed until all the instances within the dataset are used as a test instance.

For comparing TEAK with other *k*-based methods, we made use of various *k* values, since different researchers propose making use of different *k* values[1][35]. Furthermore, number of analogies to be used for estimation, i.e. *k*-value is dataset dependent and plays a decisive role in estimation accuracy. Therefore, we included a wide range of *k*-values in our experiments so as to thoroughly compare the performance of TEAK to other *k*-based methods. In our experiments, we used 5 static *k* values: 1, 2, 4, 8, 16. Also we exploited one dynamic *k* value, which is the best performing *k* value for each particular dataset. To find the best performing *k* value of a dataset, we randomly selected ten instances from the dataset as our test set. Then we chose the *k* value that yielded the best performance as the best performing *k* value for that dataset.

3.4 Performance Measures

Our main performance measure that we use in our study is magnitude of relative error (MRE), which is basically the difference between the predicted and the actual value of a project effort value. Since we want to compare the estimation accuracy of our model to other models on a per instance basis, we used MRE as our comparison method rather than using other common comparison criteria such as mean maginitude relative Error (MMRE) or median magnitude relative error (MdMRE)[32]. Calculation of MRE for the i^{th} project in a dataset is given in Equation 3.

$$MRE = \frac{|actual_i - predicted_i|}{actual_i} \tag{3}$$

Another error based performance measure we use is pred(r). Pred(r) is the prediction at level r, i.e. the percentage of predictions whose error percentage is within $\pm\ r$ percentage of the actual value. The formula for calculating pred(r) is given in Equation 4.

$$Pred(r) = \frac{100}{N} \times \sum_{i=1}^{N} \begin{cases} 1 \text{ if } MRE_i <= r \\ 0 \text{ otherwise} \end{cases} \tag{4}$$

As we have described in Section 3.3, we make 20 rounds for each dataset and in each round we use each individual project as a test instance. For 20 runs, we store the MRE values for both TEAK and other *k* values. After that we can compare performance of TEAK to any other *k*-based CBR method ($k = i, i \in 1, 2, 4, 8, 16, bestk$) on the basis of mean MRE values and mean pred(25) values.

In addition to MRE and pred(25), we also made use of win-tie-loss values to compare the performance of TEAK to other *k*-based CBR methods. While calculating win-tie-loss values, we first check if $method_i$ and $method_j$ are statistically different according to 95% Wilcoxon signed rank test in $round_k$. If they are not statistically different, then we increase tie_i and tie_j. On the other hand, if they turn out to be different, we check their mean MRE values for all test instances in $round_k$. Then the win value of method with lower mean MRE as well as the loss value of the method with higher mean MRE value is increased by one. The pseudocode for win-tie-loss calculation is given in Figure 6

```
$win_i$ = 0, $tie_i$ = 0, $loss_i$ = 0
$win_j$ = 0, $tie_j$ = 0, $loss_j$ = 0
if WILCOXON($MRE's_i$, $MRE's_i$) says they are different then
    $tie_i = tie_i + 1$;
else
    if mean($MRE's_i$) < mean($MRE's_j$) 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
```

Fig. 6: Pseudocode for Win-Tie-Loss Calculation Between Method i and j

3.5 Threats to Validity

We will address the threats to the validity of our under four categories: 1) Internal validity, 2) external validity, 3) construct validity and 4) statistical validity.

Internal validity fundamentally questions to what extent the cause-effect relationship between dependent and independent variables hold. For addressing the threats to internal validity of our results, we used 10 datasets and applied leave-one-out[33] in our experiments so that for each iteration we used a different test instance and a different train set.

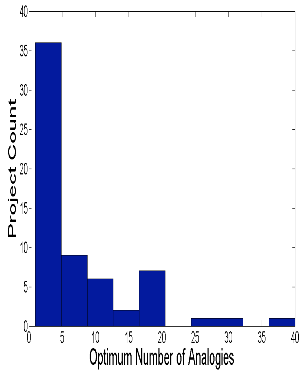
External validity, i.e. generalizability of results addresses the extent to which the findings in a particular study are applicable outside the specifications of that study[36]. To ensure the generalizability of our results, we paid extra attention to include as many datasets as possible coming from various resources and used 10 datasets from 3 different sources in our study. Our datasets contain a wide diversity of projects in terms of their sources, their domains and the time period they were developed in. Datasets composed of software development projects from different organizations around the world are used to generalize our results[6]. We also think that reproducibility of results is an important factor for external validity. Therefore, we have purposely selected publicly available datasets.

Construct validity (i.e. face validity) assures that we are measuring what we actually intended to measure[37]. In our research we are using MRE, pred(25) and win-tie-loss values for measuring and comparing performance of different models. Comparison of different models in software effort estimation domain by using error based methodologies are criticized for being unreliable[38][39]. However, a big majority of effort estimation studies use estimation-error based measures for measuring and comparing performances of different methods. We also used error based measures in our study for three reasons: 1) They are practical options for majority of researchers[40][41][3][42], 2) using error based methods enables our study to be benchmarked with previous effort estimation research and 3) MRE measures the performance of a model on individual instances, which is aligned with our datasets and experimental settings as we use leave-one-out method.

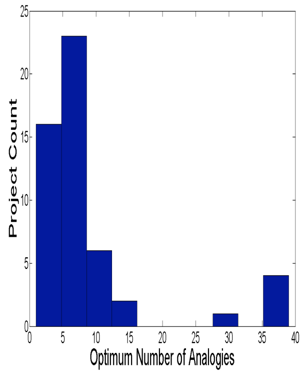
To validate our results statistically we employed statistical significance tests. To mutually compare two classifiers induced on multiple datasets, Wilcoxon signed rank test is suggested[43]. The Wilcoxon signed rank test can be viewed as the non-parametric counterpart of the *t-test*[44]. Therefore, we used Wilcoxon signed rank test in our study at a confidence level of 95%.

4 Results

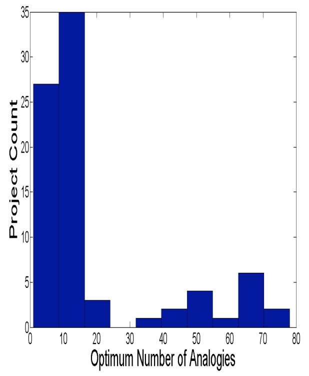
4.1 Results for Research Question 1



(a) Cocomo81



(b) Nasa93



(c) Desharnais

Fig. 7: Distribution of Optimum k Values

Our first research question was how we could better understand the underlying characteristics of dataset. Actually a majority of research studies in software effort estimation tries to address this question. However, as it was reported[34][45] most of the methods in literature were tested on a single or a very limited number datasets, thereby reducing the credibility of the proposed method. To avoid this pitfall, we included many datasets from different domains and different sources. Although k-based CBR methods have been reported to produce better results than traditional regression based methods[46][7][8] and handle datasets with discontinuities better[34], they still have limited predictive power due to the fixed number of analogies that they use. In other words, a single best performing k value that is yielding the lowest MRE values for the whole dataset does not necessarily produce lowest MRE values for individual projects. We can better see this fact from Figure 7. To produce histograms given in Figure 7, we calculated the optimum number of analogies, i.e. best-k value that yielded lowest MRE value for each project of a dataset. For a dataset of size n, optimum best-k value can range from 1 to $n-1$. What we can see from Figure 7 is that for a considerable amount of projects a fixed-k approach will be far from optimum. Since 3 datasets were enough to illustrate our viewpoint, we did not include all ten projects into Figure 7.

In this research we propose a model called TEAK that makes use of GAC trees. Unlike [illegible]ious CBR calibration methods, TEAK does not use [illegible] overall performance of the whole dataset to decide on the number of analogies, i.e. TEAK does not use a fixed number of analogies for all test instances. Instead TEAK prunes all unnecessary analogies on the basis of performance-variance for single test instance, thereby choosing the analogies to be used for each test instance dynamically. In Figure 8, we can see the box plot for the number of instances that were send to GAC2 tree. It is seen from Figure 8 that TEAK is indeed capable of selecting various k values dynamically to be used during building up GAC2 and as our results indicate dynamic instance selection that takes into account the characteristics of each dataset on the basis of performance-variance improves the prediction accuracies.

Furthermore, as we can see from Figure 9, selecting small training sets on a per-ins[illegible] basis making use of variance improves CBR effort [illegible]ation performance and outperforms fixed-k approaches.

4.2 Results for Research Question 2

The second research question was whether we could ease the procedure of selecting the best performing number of analogies and whether we could increase the predictive performance while doing that. As a matter of fact, with the utilization of GAC trees in TEAK, we have saved the researcher from the task of choosing the best k value. Since TEAK itself builds up GAC1 and prunes irrelevancies while sending the project instances to GAC2 as well as while building up GAC2, we have left the entire best k selection process to TEAK. Furthermore, apart from being able to choose the number of analogies for each test instance on its own, TEAK outperforms all the other k-based CBR methods.

When we look at the win, tie, loss values in Table 4, we see that in all ten data sets, TEAK has the highest $win - loss$ values. This suggests that TEAK has attained lower MRE values than all other methods. Indeed, TEAK has a $loss$ value of 0 for all datasets and this shows that TEAK has never been outperformed by any other method in all datasets for statistically significant cases. Also the plots of MRE lines given in Figure 9 supports the statistics in Table 4. In other words, MRE values of TEAK are lower than all other methods and therefore the MRE line for TEAK lies below all the others. Among ten datasets Albrecht seems to be the most challenging one. From the MRE lines given in Figure 9, it is difficult to claim a best performing method for Albrecht dataset. Also for pred(25) values we see a similar pattern. TEAK has the highest pred(25) value for all datasets except Albrecht. Furthermore, when compared to pred(25) values of other methods, TEAK can improve pred(25) values up to 69%.

Furthermore, three datas[illegible]ook like they challenge all methods: Albrecht, Coco[illegible]o, Nasa93c2. Regarding the remaining seven cases:

- For three cases (Cocomo81, Nasa93 and SDR) we see that TEAK is the best method by an amount of more than 90% improvement in accuracy; e.g. in SDR TEAK comes best over a hundred times compared to the next best number of $(wins - losses)$ which is 8.
- In Cocomo81, Cocomo81o, Cocomo81e, Nasa93, Desharnis TEAK comes best at least twice as much as the next contender.
- Only in one case out of seven does another method come close to TEAK: In ISBSG-Banking, TEAK comes best 54 times compared to the 44 times of $k = 16$.

5 Conclusion and Future Work

In this paper, to configure CBR systems, we have proposed a novel method called TEAK that makes use of GAC. In our study we defined two research questions to address the problems with traditional methods of configuring CBR methods: 1)Understanding the characteristics of data and 2) determining the number of analogies to be used for better performance. We have shown in Section 4 that previous methods that make use of fixed number of analogies (fixed-k) on the basis of performance score (MRE or any other error based performance measure) of the overall dataset will eventually sacrifice from the per-instance based optimum k value for a large number projects. Therefore, we proposed utilizing GAC trees that will select small training sets of different sizes, i.e. different number of analogies for each individual test instance. Furthermore, unlike traditional CBR configuration methods, we made use of performance-variance rather than performance score. Therefore, rather than proposing a best-k value a priori as the traditional CBR methods do, what TEAK does is starting with all the training samples in the dataset, learning the dataset to form GAC trees and chopping-off the irrelevant analogies on the basis of variance. As it is reported in Section 4, TEAK has outperformed previously proposed CBR configuration methods.

Apart from trying to address the reported issues regarding previous CBR studies in terms of dataset

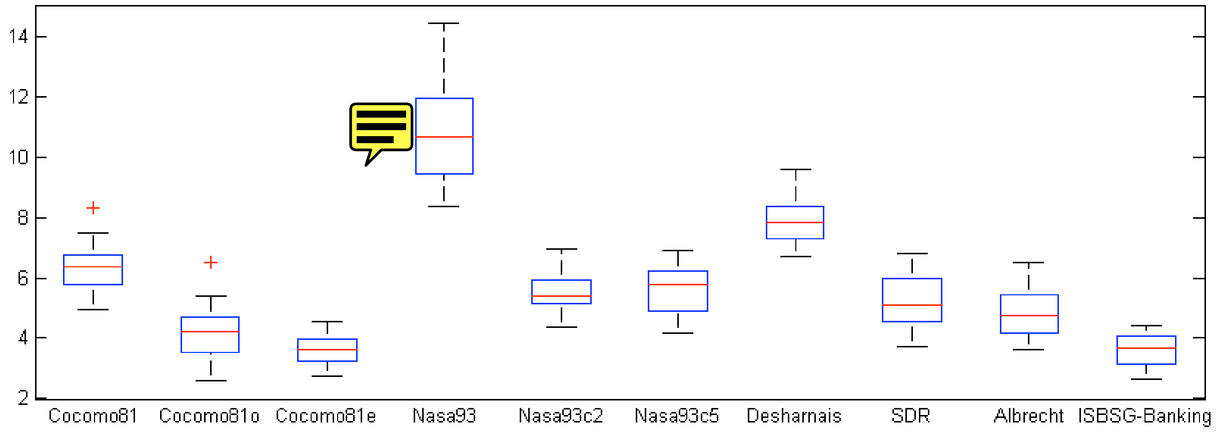


Fig. 8: Boxplot for the Number of Instances Send to GAC2 Tree

TABLE 3: All Datasets: Mean Pred(25) Values for TEAK and multiple k values

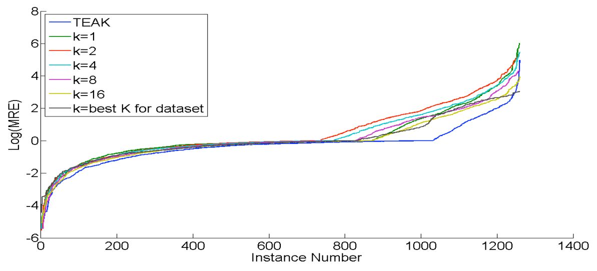
Dataset	TEAK	k=1	k=2	k=4	k=8	k=16	k=best K
Cocomo81	0.13	0.09	0.09	0.09	0.1	0.11	0.1
Cocomo81e	0.24	0.16	0.15	0.19	0.18	0.24	0.2
Cocomo81o	0.14	0.07	0.08	0.1	0.1	0.1	0.1
Nasa93	0.15	0.1	0.12	0.11	0.11	0.11	0.11
Nasa93c2	0.17	0.1	0.11	0.12	0.11	0.14	0.1
Nasa93c5	0.21	0.11	0.11	0.15	0.17	0.18	0.19
Desharnais	0.3	0.17	0.2	0.24	0.26	0.27	0.26
SDR	0.22	0.13	0.06	0.1	0.08	0.09	0.12
Albrecht	0.15	0.15	0.23	0.21	0.23	0.26	0.22
ISBSG-Banking	0.33	0.19	0.18	0.23	0.26	0.3	0.23

and model, we also tried to meet the methodological problems such as testing only on a limited number of datasets[1] and lacking statistical checks on the results[34]. Therefore, we utilized various datasets from multiple resources and evaluated our results on the basis of Wilcoxon signed rank test at a 95% confidence level.

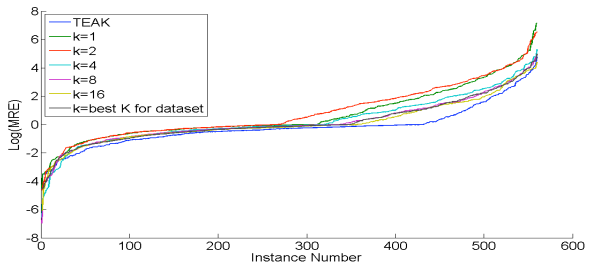
Going forward, we would like to see how the removal of irrelevant instances in TEAK via utilization of GAC trees relate to the analogies that produce the lowest MRE for each individual instance. That is, we would like to see the relationship between train instances that produced the histograms of Figure 7 and the train instances that are selected by TEAK. We believe that the MRE values achieved by using the optimum number of analogies for each test instance forms a baseline for k-based CBR methods and in our study we have observed that we obtained closer re[illegible] to this baseline than any other method. However, there is still space to discover between the MRE values of TEAK and the baseline. As our future work we will try to discover that space and lower the MRE values closer to the baseline.

REFERENCES

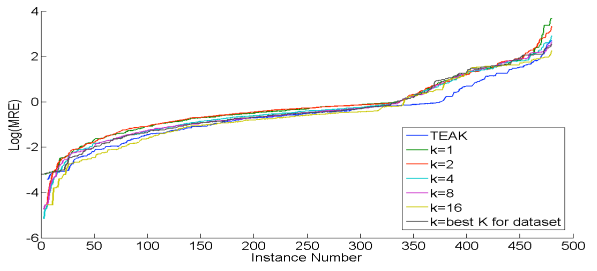
[1] G. Kadoda, M. Cartwright, and M. Shepperd, "On configuring a case-based reasoning software project prediction system," *UK CBR Workshop, Cambridge, UK*, pp. 1–10, 2000.
[2] F. Heemstra, "Software cost estimation," *Information and Software Technology*, pp. 1–14, 1992.
[3] T. Menzies, Z. Chen, J. Hihn, and K. Lum, "Selecting Best Practices for Effort Estimation," *IEEE Transactions on Software Engineering*, vol. 32, pp. 883–895, 2006.
[4] M. Jø rgensen, "A review of studies on expert estimation of software development effort," *Journal of Systems and Software*, vol. 70, pp. 37–60, February 2004.
[5] 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.
[6] *A new perspective on data homogeneity in software cost estimation: a study in the embedded systems domain*, 2009. [Online]. Available: http://dx.doi.org/10.1007/s11219-009-9081-z
[7] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Trans. Softw. Eng.*, vol. 23, no. 11, pp. 736–743, 1997.
[8] M. Shepperd, C. Schofield, and B. Kitchenham, "Effort estimation using analogy," pp. 170–178, 1996.
[9] B. W. Boehm, *Software Engineering Economics*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.
[10] B. W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, R. Madachy, and B. Steece, *Software Cost Estimation with Cocomo II*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.
[11] P. Kruchten, "The Rational Unified Process: an introduction," 1999.
[12] T. Menzies, O. Elrawas, J. Hihn, M. Feather, R. Madachy, and B. Boehm, "The business case for automated software engineering," pp. 303–312, 2007.
[13] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *IEEE Trans. Softw. Eng.*, vol. 21, no. 2, pp. 126–137, 1995.
[14] G. R. Finnie and G. E. Wittig, "Ai tools for software development effort estimation," p. 346, 1996.
[15] M. Jorgensen and M. Shepperd, "A systematic review of software



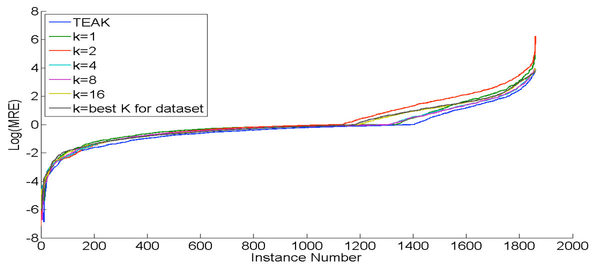
(a) Cocomo81



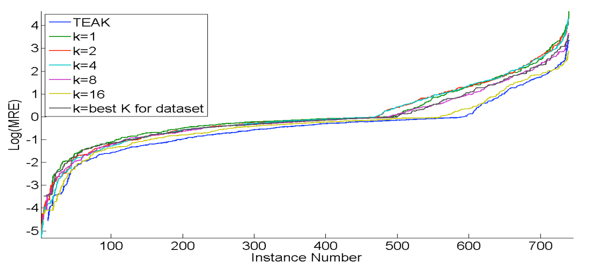
(b) Cocomo81e



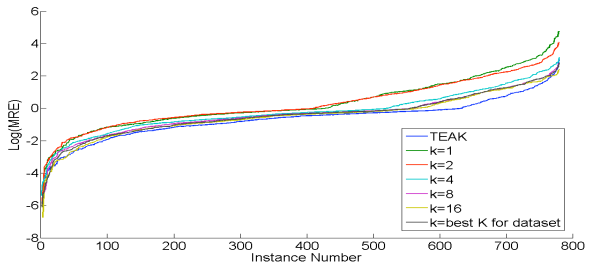
(c) cocomo81o



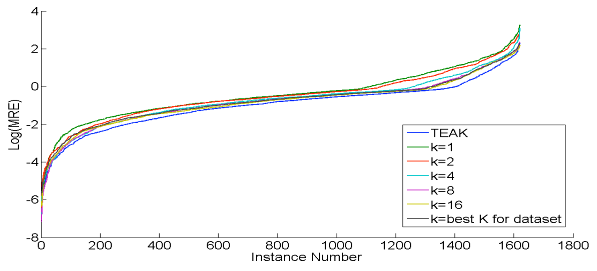
(d) Nasa93



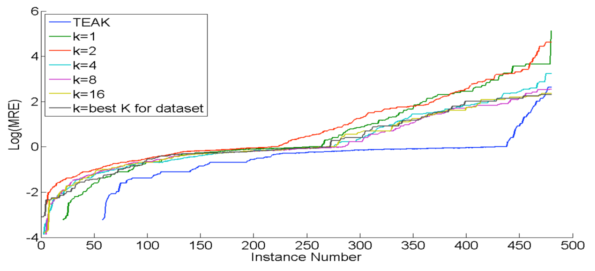
(e) Nasa93c2



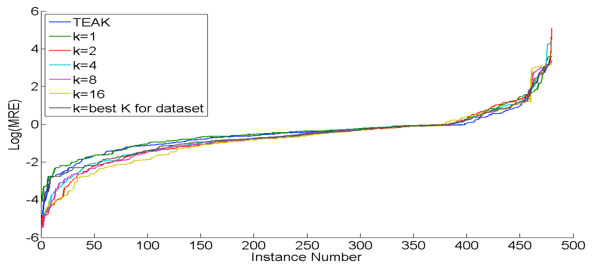
(f) Nasa93c5



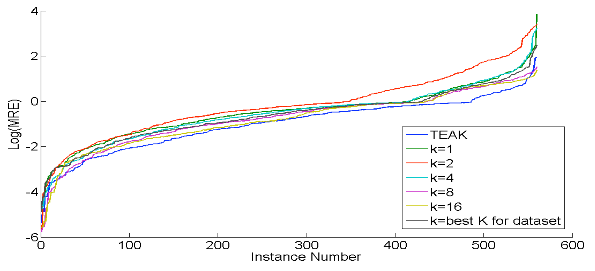
(g) Desharnais



(h) SDR



(i) Albrecht



(j) ISBSG

Fig. 9: Log-Mre Values of Datasets for GAC2 and multiple k Values

TABLE 4: Win-Tie-Loss Values for TEAK and multiple k values

(a) Cocomo81

Algorithm	Win	Tie	Loss	Win - Loss
TEAK	108	132	0	108
k=1	28	148	64	-36
k=2	16	136	88	-72
k=4	20	160	60	-40
k=8	34	170	36	-2
k=16	42	154	44	-2
k=bestK	52	180	8	44

(b) Cocomo81e

Algorithm	Win	Tie	Loss	Win - Loss
TEAK	110	130	0	110
k=1	20	134	86	-66
k=2	6	118	116	-110
k=4	26	166	48	-22
k=8	34	168	38	-4
k=16	44	178	18	26
k=bestK	72	162	6	66

(c) Cocomo81o

Algorithm	Win	Tie	Loss	Win - Loss
TEAK	22	218	0	22
k=1	2	196	42	-40
k=2	2	222	16	-14
k=4	8	228	4	4
k=8	8	232	0	8
k=16	12	228	0	12
k=bestK	8	232	0	8

(d) Nasa93

Algorithm	Win	Tie	Loss	Win - Loss
TEAK	80	160	0	80
k=1	12	122	106	-94
k=2	18	132	90	-72
k=4	28	178	34	-6
k=8	40	182	18	22
k=16	38	194	8	30
k=bestK	44	192	4	40

(e) Nasa93c2

Algorithm	Win	Tie	Loss	Win - Loss
TEAK	42	198	0	42
k=1	16	176	48	-32
k=2	2	166	72	-70
k=4	18	200	22	-4
k=8	28	208	4	24
k=16	26	206	8	18
k=bestK	28	206	6	22

(f) Nasa93c5

Algorithm	Win	Tie	Loss	Win - Loss
TEAK	48	192	0	48
k=1	2	172	66	-64
k=2	6	182	52	-46
k=4	12	200	28	-16
k=8	26	208	6	20
k=16	32	208	0	32
k=bestK	28	210	2	26

(g) Desharnais

Algorithm	Win	Tie	Loss	Win - Loss
TEAK	64	176	0	64
k=1	2	140	98	[illegible]
k=2	14	156	70	[illegible]
k=4	32	192	16	16
k=8	30	202	8	22
k=16	34	200	6	28
k=bestK	30	202	8	22

(h) SDR

Algorithm	Win	Tie	Loss	Win - Loss
TEAK	102	138	0	102
k=1	20	176	44	-24
k=2	2	170	68	-66
k=4	16	202	22	-6
k=8	26	196	18	8
k=16	18	188	34	-16
k=bestK	22	198	20	2

(i) Albrecht

Algorithm	Win	Tie	Loss	Win - Loss
TEAK	6	234	0	6
k=1	0	230	10	-10
k=2	0	234	6	-6
k=4	0	240	0	0
k=8	4	236	0	4
k=16	6	234	0	6
k=bestK	0	240	0	0

(j) ISBSG-Banking

Algorithm	Win	Tie	Loss	Win - Loss
TEAK	54	186	0	54
k=1	6	154	80	-74
k=2	8	154	78	-70
k=4	18	194	28	-10
k=8	36	196	8	28
k=16	44	196	0	44
k=bestK	34	200	6	28

development cost estimation studies," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 33–53, 2007.

[16] T. Menzies and J. Hihn, "Evidence-based cost estimation for better-quality software," *IEEE Softw.*, vol. 23, no. 4, pp. 64–66, 2006.

[17] T. Babu and M. Murty, "Comparison of genetic algorithm based prototype selection schemes," *Pattern Recognition*, vol. 34, p. 523525, 2001.

[18] Y. Huang, C. Chiang, J. Shieh, and E. Grimson, "Prototype optimization for nearest-neighbor classification," *Pattern Recognition*, vol. 35, p. 12371245, 2002. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0031320301001248

[19] U. Lipowezky, "Selection of the optimal prototype subset for 1-NN classification," *Pattern Recognition Letters*, vol. 19, p. 907918, 1998. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167865598000750

[20] C. Kirsopp and M. Shepperd, "Making inferences with small numbers of training sets," *IEE Proc.*, vol. 149, 2002.

[21] 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.

[22] 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.

[23] D. Beeferman and A. Berger, "Agglomerative clustering of a search engine query log," *In Knowledge Discovery and Data Mining*, vol. pages, pp. 407–416, 2000.

[24] S. Guha, R. Rastogi, and K. S. Cure, "An efficient clustering algorithm for large databases," *In In Proceedings of ACM SIGMOD International Conference on Management of Data*, vol. pages, pp. 73–84, 1998.

[25] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein, "Cluster

Analysis and Display of Genome-Wide Expression Patterns. Proc. of the National Academy of," *Science*, vol. 95, pp. 14 863–14 868.

[26] B. Walter, K. Bala, M. Kulkarni, and K. Pingali, "Fast agglomerative clustering for rendering," *IEEE Symposium on Interactive Ray Tracing (RT)*, pp. 2–7, 2008.

[27] G. Boetticher, T. Menzies, and T. Ostrand, "PROMISE repository of empirical software engineering data," 2007. [Online]. Available: http://promisedata.org/repository

[28] T. Menzies, D. Port, Z. Chen, J. Hihn, and S. Stukes, "Validation methods for calibrating software effort models," pp. 587–595, 2005.

[29] 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.

[30] F. Walkerden and R. Jeffery, "An empirical study of analogy-based software effort estimation," *Empirical Softw. Engg.*, vol. 4, no. 2, pp. 135–158, 1999.

[31] C. Lokan, T. Wright, P. R. Hill, and M. Stringer, "Organizational benchmarking using the isbsg data repository," *IEEE Softw.*, vol. 18, no. 5, pp. 26–32, 2001.

[32] A. Bakir, "Classification based cost estimation model for embedded software," Master's thesis, Bogazici University, 2008.

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

[34] B. Kitchenham and E. Mendes, "Why comparative effort prediction studies may be invalid," pp. 1–5, 2009.

[35] L. C. Briand, "A Pattern Recognition Approach for Software Engineering Data Analysis," *IEEE Transactions on Software Engineering*, vol. 18, pp. 931–942, 1992.

[36] D. Milic and C. Wohlin, "Distribution Patterns of Effort Estimations," in *Euromicro*, 2004.

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

[38] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion MMRE," *IEEE Transactions on Software Engineering*, vol. vol, pp. 29no11pp985–995.

[39] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Transactions on Software Engineering*, vol. vol, pp. 31no5pp380–391.

[40] R. Premraj and T. Zimmermann, "Building Software Cost Estimation Models using Homogenous Data," *ESEM*, 2007.

[41] K. Lum, T. Menzies, and D. Baker, "2CEE, A TWENTY FIRST CENTURY EFFORT ESTIMATION METHODOLOGY," *ISPA / SCEA*, pp. 12 – 14, 2008.

[42] 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

[43] J. Demsar, "Statistical comparisons of classi ers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, 2006.

[44] Y. Jiang, B. Cukic, T. Menzies, and N. Bartlow, "Comparing Design and Code Metrics for Software Quality Prediction," pp. 11–18, 2008.

[45] B. A. Kitchenham, L. M. Pickard, S. G. Macdonell, and M. J. Shepperd, "What accuracy statistics really measure," *IEEE Proceedings-Software*, vol. 148, pp. 101 049/ip–sen20 010 506, 2001.

[46] K. Atkinson and M. J. S. ', "The use of function points to find cost analogies'," *in Proc. European Software Cost Modelling Meeting . Ivrea, Italy:*, 1994.

PLACE PHOTO HERE

Ekrem Kocaguneli Biography text here.

Tim Menzies Biography text here.

Ayse Bener Biography text here.

Jacky W. Keung Biography text here.