Exploiting the Essential Assumptions of Analogy-based Effort Estimation

Vivek Gopalakrishnan
North Carolina State University
vgopala2@ncsu.edu

Muthu Arvind Lakshmanan
North Carolina State University
mlakshm@ncsu.edu

ABSTRACT

There are many design options for software effort estimators. We identified the essential assumption of analogy-based effort estimators i.e. projects with similar product and project features have similar effort estimates. We test this assumption by creating a cluster tree and comparing the sub-trees with the super-trees using variance as a metric. We then compare various pruning and search mechanisms to find the effort estimates.

Keywords

Software effort estimation, Analogy based estimation, kNN, TEAK

1. INTRODUCTION

Software effort estimates are often wrong by a factor of four [1] or even more [2]. As a result, the project might be affected due to insufficient resources. In the worst case scenario, the project gets cancelled as in the case of NASA which canceled the Check-out Launch Control System project when the intiail estimate of $200M was increased to $400M [6].

We need better ways to generate accurate project effort estimates. In this paper, we explore the various analogy-based estimators (ABE) and TEAK which stands for Test Essential Assumption Knowledge[3]. The principle for designing an effort estimator based on TEAK is:

> *Find the situations that confuse estimation. Remove those situations.*

In this paper, we compare the different effort estimators built based on ABE and TEAK and evaluate them. THe rest of this paper is structured as follows: Section 2 is used to review about the different effort estimators, Section 3 is used to discuss about TEAK, Section 4 compares the performance of the learners using the various error metrics. Our conclusion is that no learner is perfect for all the datasets.

The paper uses the notation of Table 1.

2. BACKGROUND

2.1 Scope

This project is a detailed comparison of various analogy-based effort estimation learners along with linear regression

Symbol	Explanation
ABE	Analogy based Estimation
ABE0	A baseline ABE method
NNet	A neural net prediction system with one hidden layer
LR	Linear regression
k-NN	k Nearest Neighbors learner
k	Selected similar projects
GAC	Greedy Agglomerative Clustering
GAC1, GAC2	First and second GAC trees within TEAK
TEAK	Test Essential Assumption Knowledge
TRAVERSE	Tree traversal algorithm for finding the effort estimate
TRAVERSE2	
TEAK0	TEAK with no pruning
TEAK2	Modifications of TEAK
X, Y	2 instances/projects in a dataset
x, y	Two vertices in GAC tree
$X_{\mathrm{i}}, Y_{\mathrm{i}}$	i^{th} features of projects X and Y
$L_{\mathrm{x}}, L_{\mathrm{y}}, L_{\mathrm{z}}$	Leaves of sub-trees whose roots are x, y and z
$k_{\mathrm{x}}, k_{\mathrm{y}}, k_{\mathrm{z}}$	The number of leaves in $\mathrm{L_x}$, $\mathrm{L_y}$ and $\mathrm{L_z}$
$\sigma^2{}_{\mathrm{x}}$	Variance of instances in x
$\sigma^2{}_{\mathrm{yz}}$	Weighted sum of variances of y and z
γ	User-defined value for pruning GAC
R	Random number between 0 and 1
$max(\sigma^2)$	Maximum variance of all sub-trees in a GAC tree
T	Given dataset
N	Test set out of the dataset T
$actual_i$	The actual effort seen in test instance $N_i \in N$
$predicted_i$	The effort of test instance $N_i \in N$ predicted by some prediction system
AR	Absolute residual $\lvert x_i - \hat{x}_i \rvert$
MRE	Magnitude of relative error $\frac{AR}{actual_i}$
PRED(X)	The percentage of estimates that are within X% of the actual value

Table 1: The explanations of symbols that are used in this paper. Symbols that are related to each other are grouped together.

and neural networks. This comparison is made to ensure that the performance of analogy based effort estimators are comparable to the other methods in widespread use.

Analogy-based effort estimation is a widely-used and studied technique. The TEAK learner is a novel method to improve that technique. It removes the confusion in the data by pruning all subsets of data with high variance. The idea behind this heuristic is that if a test instance falls into subsets with high variance, any minor change in the test instance will lead to a major change in the effort estimate for the test instance. By removing these subsets, effort estimation by analogy might be improved.

Proposals similar to TEAK have been found in requirements engineering. In the paper "To Be and not To Be"[5], they discuss a range of methods for handling such inconsistencies. One method is *circumvent* which adds "pollution markers" that screens problematic regions away from the rest of the system. TEAK uses a similar method by comparing variance since it guides the reasoning away from the problematic training data , i.e., the high varying sub-trees.

2.2 Effort Estimation

There are various effort estimation method currently in use:

- Algorithmic methods/parametric methods
- Induced prediction systems
- Human-centric techniques

While human-centric techniques are the most widely used method, it is problematic. Algorithmic methods and induced prediction system. COCOMO is a widely used parametric method. Boehm's 1981 COCOMO model hypothesized that effort was exponential on Lines of Code (LOC) and linear on 15 effort multipliers. Induced prediction systems are better in certain cases where the local training data does not conform to the parametric model's requirements.

2.3 Non-analogy based methods

There are many induction methods like linear regression, neural networks and analogy based estimators. Linear regression is useful when the data fits some function. The parameters of the function are then adjusted in order to reduce the difference between the actual and predicted effort estimations. An example of a linear regression model is :

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...$$

where x_i are model input features, y is the model output and β_i are the coefficients/model parameters adjusted be the linear regression system.

The issue with linear regression is that it assumes the data is linearly separable. But this is not possible in all the cases. Various patches have been proposed where we take logarithms of exponential distributions before applying linear regression.

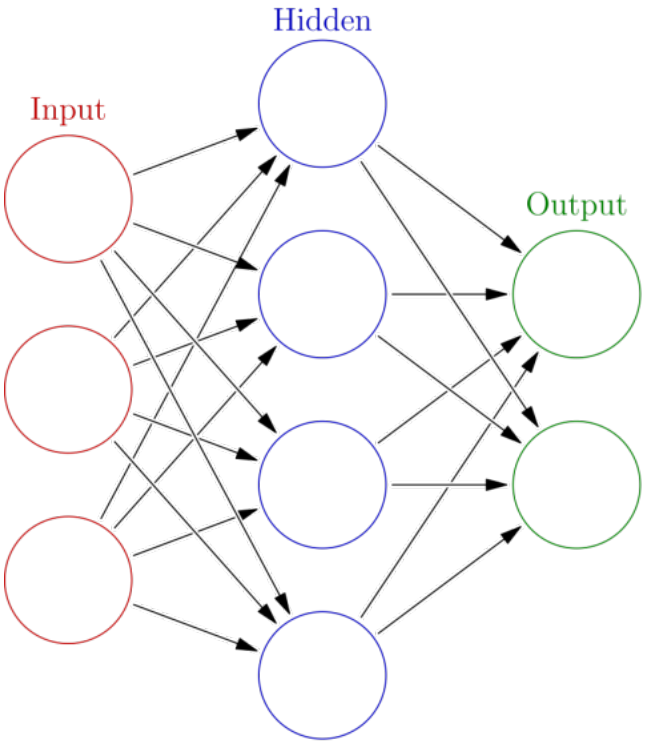

Figure 1: A sample neural network containing 1 hidden layer with 4 nodes, 3 nodes in the input layer and 2 nodes in the output layer.

Neural nets are useful when the data distributions are not simple linear functions. The input layer consists of project details which is connected to zero or more hidden layers which is connected to the output node which is the effort prediction. The connections are weighted directed edges connecting the nodes. If the sum of the signals arriving to a node is greater than a threshold value, the node fires and a weight is propagated across the network. The output is compared to the expected effort value and then a correction is applied as necessary. 1 shows a sample neural network.

2.4 Analogy-based Estimation

The basic principle of Analogy-based estimation is that:

> *Projects that are similar with respect to project and product factors will be similar with respect to project effort.*

We used a baseline ABE called ABE0 as follows:

- Each row represents one project
- Columns contain independent variables in projects and the dependent variables are the effort required to complete one project.

After processing the training projects, ABE0 outputs an estimate when a test project is given. A scaling measure is

used to ensure that all independent features have the same degree of influence on the distance measure between test and training projects.

The similarity between the the test project and each project in the dataset is calculated using a similarity measure. A similarity measure measures the closeness or the distance between two data points in an n-dimensional feature space. The result is usually represented in a distance matrix identifying the similarity among all the cases in the dataset. The commonly used distance metric is the Euclidean distance metric. It is based on the principle of Pythagorean Theorem to derie a straight line distance between two points in n-dimensional space.

In general, the unweighted Euclidean distance between two points $P = (p_1, p_2, ..., p_n)$ and $Q = (q_1, q_2, ..., q_n)$ and can be defined and calculated as :

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \quad (1)$$

An alternative is to apply different weights to each individual project feature to reflect its relative importance in the prediction system. The weighted Euclidean distance can be calculated as :

$$\sqrt{w_1(p_1 - q_1)^2 + w_2(p_2 - q_2)^2 + ... + w_n(p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} w_i(p_i - q_i)^2} \quad (2)$$

where w_1 and w_n are the weights of the 1^{st} and n^th project features.

For the purpose of this project, we have used the unweighted Euclidean distance metric to calculate the similarity between two projects. There are 5 variants of ABE0 implemented differing in the values of k. The package implements k = {1, 2, 4, 8, 16}

3. TEAK

The basic principle of TEAK is to remove the confusion present in the training set and then build the learner.

3.1 Training

TEAK is an ABE0 with the following variations:

3.1.1 Select a prediction system

The first step is to select a prediction system for TEAK. We use ABE as the prediction system since:

- It is widely studied
- It works even if the domain data is sparse
- It makes no assumptions about the data distributions or an underlying model
- When the local data doesn't support standard parametric methods, ABE can still be applied.

3.1.2 Identify the essential assumption(s)

The basic hypothesis underlying the analogy-based effort estimation is that projects that are similar with respect to project and product factors will be similar with respect to project effort. The issue with this method is that projects from a high variance region are likely to have very different project effort values and can decrease the accuracy in estimation.

Let us assume that a node x has a left child y and a right child z. Going from the parent node x to child nodes will create 2 cases:

- The variance decreases when the number of instances decreases.
- The variance increases when the number of instances decreases.

Hence, it is not necessarily true that moving to as smaller set of neighbors decreases variance.

3.1.3 Identify Assumption Violation

The third step is to recognize situations that violate the essential assumption. This step requires comparing the variance of larger-k estimates to smaller-k estimates. The method used to achieve this is to build a cluster tree where each sub-tree contains training data that is closer together than the super-tree.

In this project, we use a simple method called greedy agglomerative clustering (GAC). GAC is a bottom-up clustering algorithm that groups nodes together from level i that are the closest pairs to form the higher level *(i + 1)*. The algorithm terminates when there is only one node at some level. It is a greedy algorithm since it never backtracks looking for better pairings at level i to reduce the distance between the distance between nodes at level *i + 1*.

The final output of GAC is a tree like 2. Each leaf node represents a data row in the training data. All the other nodes are generated by the GAC algorithm. They represent the median of pairs of leaves, the medians of this median, and so on. It is a recursive algorithm which stops when there is only one node left. A GAC tree is a tree of clusters where each node at height i is the centroid of the sub-clusters at height *i + 1*. The maximum height of a GAC tree is $log_2(T)$ where T is the number of instances available. The number of nodes is halved at each level while building a GAC tree. Hence, the total number of distance calculations required for building the GAC tree is:

$$\left(\sum_{i}^{log_2 T} \left(\frac{T}{2^{i-1}} \right)^2 \right) = \frac{4}{3} \left(T^2 - 1 \right)$$

Hence, it is a $O(T^2)$ computation.

3.1.4 Remove those situations

The fourth step in TEAK's design is to remove the situations that violate the essential assumption. A sub-tree usually has

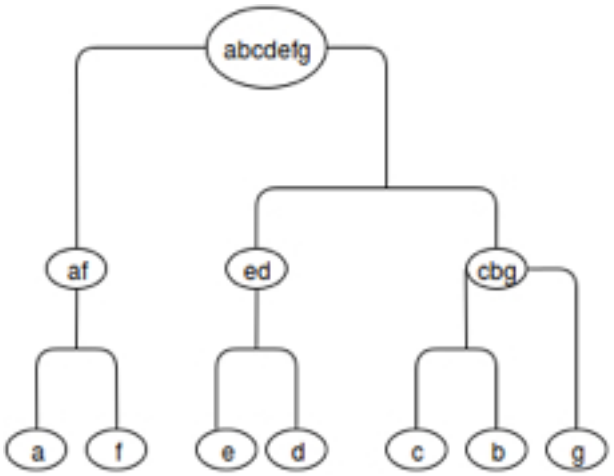

Figure 2: A sample GAC tree built from 7 instances. At each level, the two closest nodes are paired up to create the nodes in the higher level. If odd number of nodes are present, the left over node is merged with the closest node in the higher level. In this example, *g* is paired up with *cb*.

a higher variance compared to the super-tree if the super-tree contains mostly similar effort values but one sub-tree has a minority of outliers.

A sub-tree pruning policy is used in the project to prune sub-trees with a variance that violates the essential assumption. We used the policy from Kocaguneli[3] that removed sub-trees is they had:

> more than $R^{\gamma} * max\left(\sigma^2\right)$, where R is a random number $0 \leq R \leq 1$.

On an average, around 10% of the projects were retained by the pruning algorithm.

3.1.5 *Execute the modified prediction system*

The final step in the design of TEAK is to build a new prediction system. The baseline model for TEAK executes as follows:

- Apply GAC to the training dataset to build a tree, GAC1.
- Prune GAC1 using the sub-tree pruning policy described in the previous section. The remaining leaf nodes are the prototypes to be used in effort estimation.
- Apply GAC to the remaining leaf nodes or the prototypes to build a second tree, GAC2
- Compute the estimate for a test project from the median value of the projects found using the TRAVERSE algorithm described in the next section.

3.2 Prediction System

Using the GAC tree, we can find the k-nearest neighbors in the project data using the TRAVERSE procedure below:

1. Place the test project at the root of the GAC tree.
2. Move the test project to the nearest child. The distance is calculated using the formula defined by Equation 1
3. Stop if the number of projects in the node is lesser or equal to k. Else, go to step 2.

A $k = N$ nearest neighbor estimate comes from TRAVERSE-ing to a sub-tree with N leaves. Specifically, a $k = 1$ nearest neighbor estimate comes from TRAVERSE-ing to a leaf. In case of $k > 1$ nearest neighbor, the estimate is calculated as the median of the effort of the leaves.

TRAVERSE can help in testing the essential assumption of TEAK i.e., moving from a parent to a child in the GAC tree reduces the variance.

- The sub-trees starting at *x, y, z* have leaves L_x, L_y, L_z. Also, $L_z = L_x \cup L_y$.
- The number of leaves in sub-tree starting at x is $k_x = k_y + k_z$
- The variance of leaves' efforts are $\sigma_x^2, \sigma_y^2, \sigma_z^2$
- The variance of the trees below x denoted by σ_{yz}^2 is the weighted sum :

$$\sigma_{yz}^2 = \frac{k_y}{k_x}\sigma_y^2 + \frac{k_z}{k_x}\sigma_z^2$$

Parent trees have the nodes of their children plus one. If we TRAVERSE from a parent x to a child, then the sub-tree size k decreases. That is, TRAVERSE-ing moves into smaller sub-trees at each iteration. The essential assumption holds if moving from a parent node x to children y and z decreases the variance. That is:

$$\forall x \in T : \sigma_x^2 > \sigma_{yz}^2 \tag{3}$$

3.2.1 *TRAVERSE2*

TRAVERSE2 is a modification of TRAVERSE. Using a GAC tree, we can find the required projects in the training data using the following procedure:

1. Place the test project at the root of the GAC tree.
2. Move the project to the nearest child if the sub-tree variance is less than the current node variance. Else, return the projects represented by the current node. The distance is calculated using the formula defined by Equation 1.
3. Go to step 2

TRAVERSE2 ensures that the essential assumption is never violated.

3.3 Variations of TEAK

We implements 5 variants of the baseline TEAK. Let's call the baseline TEAK as TEAK9. For TEAK9, we used $\gamma = 9$.

The first variant of TEAK9 is TEAK4.5 where we used $\gamma = 4.5$. Decreasing the value of γ decreases the number of projects retained. We wanted to check the effect of decreasing γ on the performance of TEAK.

The second variant of TEAK9 is TEAK2.2 where we used $\gamma = 2.2$.

For both TEAK4.5 and TEAK2.2, the complete process can be described as :

- Apply GAC to the training dataset to build a tree, GAC1.
- Prune GAC1 using the sub-tree pruning policy with $\gamma = 4.5$ for TEAK4.5 and $\gamma = 2.2$ for TEAK2.2. The remaining leaf nodes are the prototypes to be used in effort estimation.
- Apply GAC to the remaining leaf nodes or the prototypes to build a second tree, GAC2
- Compute the estimate for a test project from the median value of the projects found using the TRAVERSE algorithm.

The third variant of TEAK was TEAK0 which had no pruning. This variant was used to see the performance of TEAK without pruning. The process for TEAK0 can be described as :

- Apply GAC to the training dataset to build a tree, GAC1.
- Compute the estimate for a test project from the median value of the projects found using the TRAVERSE algorithm.

The fourth variant of TEAK was TEAK2 which uses TRAVERSE2 for the prediction. Like TEAK0, TEAK2 also has no pruning since a similar effect is achieved by using TRAVERSE2 which ensures that the essential assumption is always satisfied.

- Apply GAC to the training dataset to build a tree, GAC1.
- Compute the estimate for a test project from the median value of the projects found using the TRAVERSE2 algorithm.

4. COMPARISONS

This section compares the performance of the induced prediction systems implemented in the project including the variants of TEAK, ABE0 along with linear regression and neural nets. To compare the performance,

- we performed a 5x5 cross-validation
- used 3 performance metrics - AR, MRE and PRED(25)

4.1 Randomized Trials

Since TEAK has a random parameter associated with it in the pruning step, it is better to repeat the evaluations multiple times before concluding. We performed a 5x5 cross validation where for each dataset, we repeated the following steps 5 times:

- Randomize the order of the rows in the current dataset.
- Perform a 5-way cross-validation
- Calculate the performance measures - AR, MRE and PRED(25)
- Rank the learners using Scott-Knott analysis.

In 5-way cross-validation, the dataset T is divided into 5 bins. When a bin_i is used for testing, the remaining $T - bin_i$ bins are used for training. Even though we used the 5x5 cross-validation, the package also supports Leave-One-Out cross-validation.

We did not do any feature subset selection for the purpose of this project. Table **??** shows the statistics about the datasets used in this project.

4.2 Details

For each of the datasets, we generated the estimates using the following learners:

- 5 variants of ABE0 which returned the median effort seen in the k-th nearest neighbors for $K \in \{1, 2, 4, 8, 16\}$
- 5 variants of TEAK - TEAK0, TEAK2, TEAK2.2, TEAK4.5, TEAK9
- Neural Net
- Linear Regression

Once the efforts were estimates, we used three performance metrics - AR, MRE, PRED(25) and used them to rank the learners. Error measures comment on the success of a prediction.

The first metric is absolute residual (AR). The magnitude of the absolute residual is the difference between the predicted and the actual values:

$$AR = |actual_i - predicted_i| \tag{4}$$

where $actual_i$ is the actual value and $predicted_i$ is the predicted value.

The second metric is magnitude of relative error (MRE). MRE is a widely used evaluation criterion for selecting the best effort estimator from a number of competing software prediction models [4]. MRE measures the error ratio between the actual effort and the predicted effort. It can be expressed as:

$$MRE_i = \frac{|actual_i - predicted_i|}{actual_i} \tag{5}$$

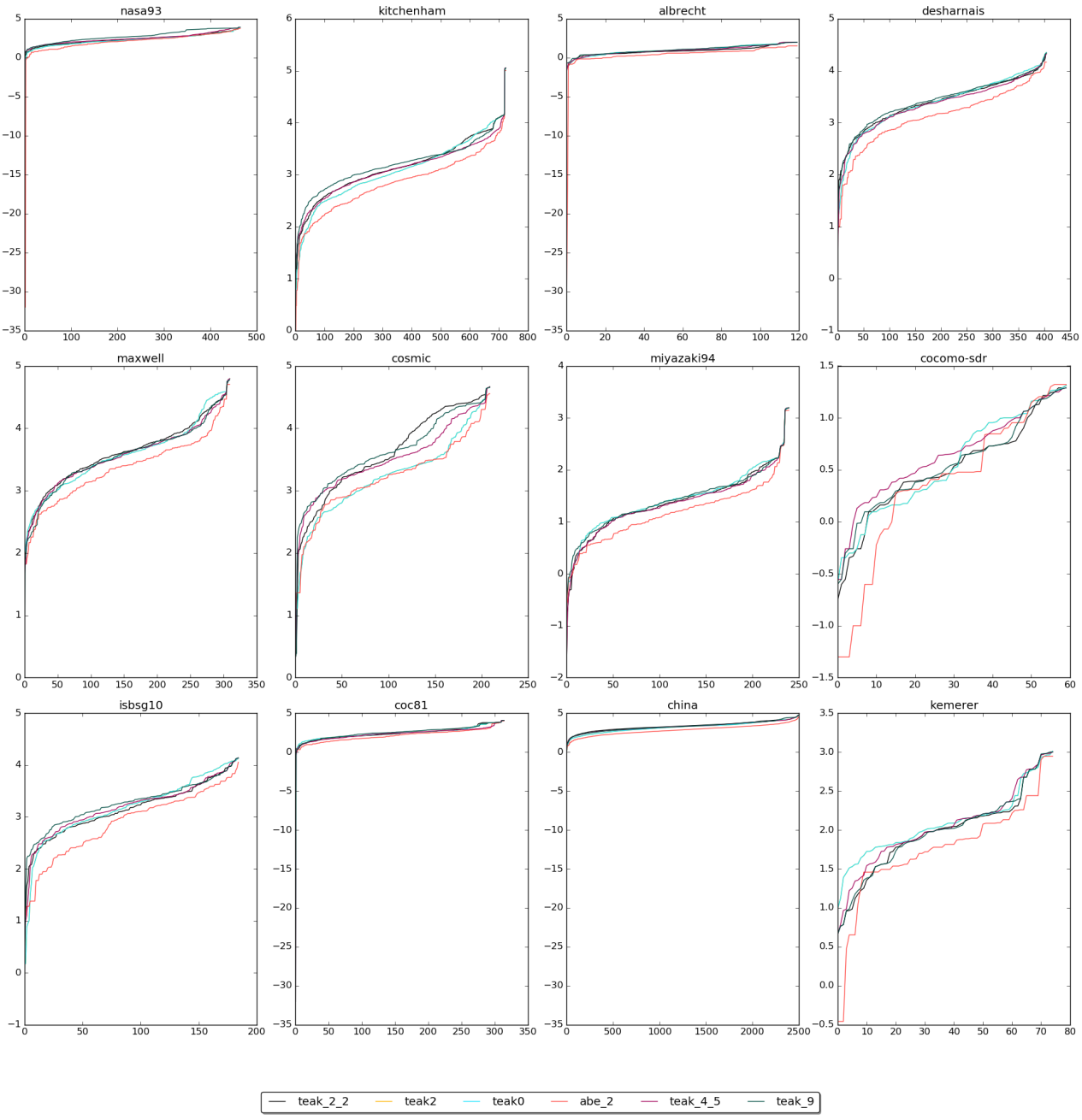
nasa93
kitchenham
albrecht
desharnais
maxwell
cosmic
miyazaki94
cocomo-sdr
isbsg10
coc81
china
kemerer
teak_2_2
teak2
teak0
abe_2
teak_4_5
teak_9

Figure 3:

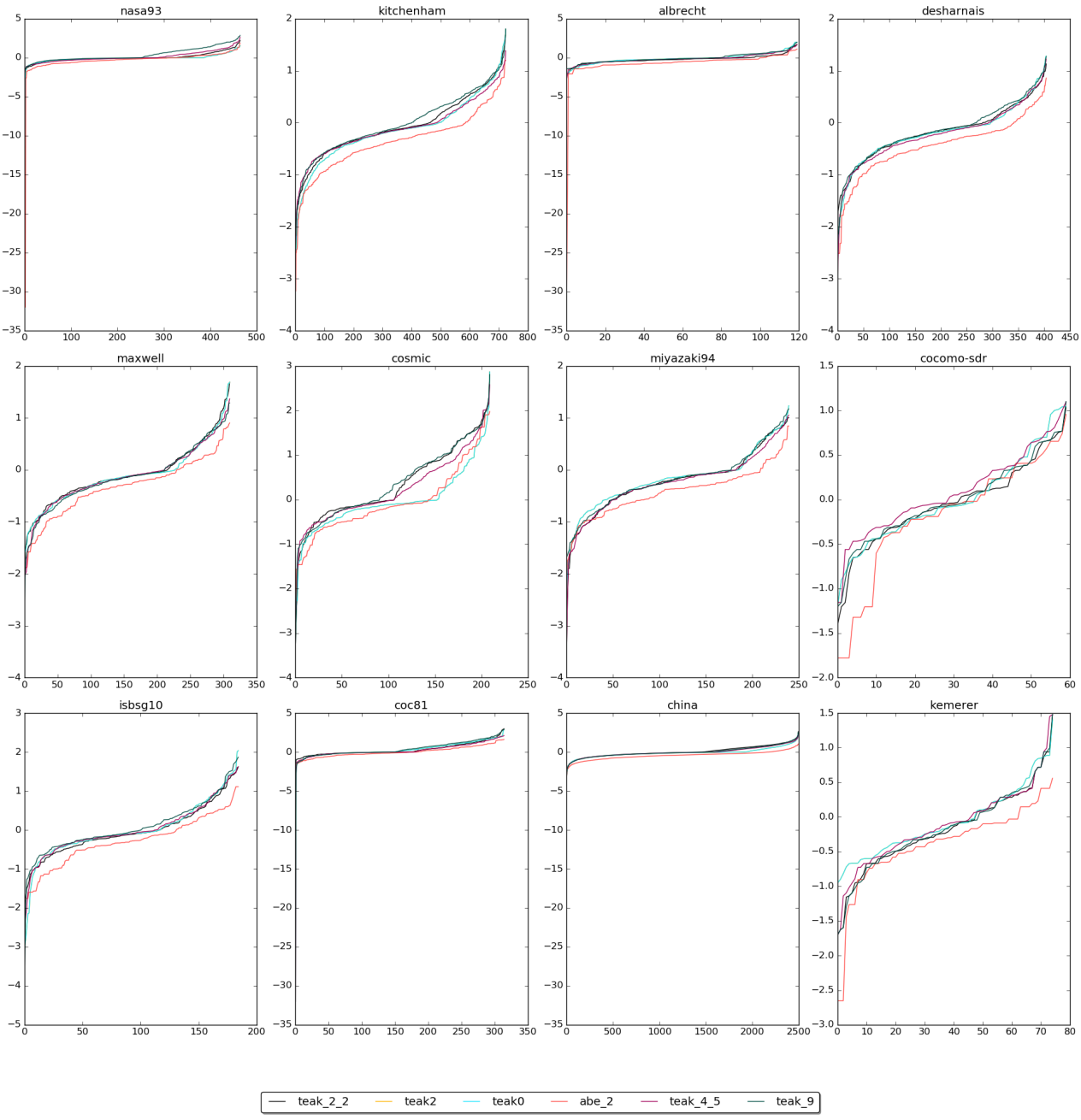
nasa93
kitchenham
albrecht
desharnais
maxwell
cosmic
miyazaki94
cocomo-sdr
isbsg10
coc81
china
kemerer
teak_2_2
teak2
teak0
abe_2
teak_4_5
teak_9

Figure 4:

The final performance metric used is PRED(X). PRED(X) reprots the average percentage of the N estimates in the test set that are within X% of the actual values. For example, PRED(25) = 50% means that 50% of the estimates were within 25% of the actual value. PRED(X) can be calculated using:

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

A notable difference among these metrics is that AR and MRE are calculated for every item in a test set while PRED(X) is a summary statistic that reports the behavior over an entire test suite.

Once the error metrics were calculated, the learners were ranked using a statistical test - Scott Knott. For the performance metrics AR and MRE, the median of the error values were computed for each run in the 5x5 cross-validation and the 25 resultant values were used to compute the ranks. In case of PRED(25), since it is a summary statistic which generates 1 value per test suite, the 25 values obtained totally from each of the test suite were used to rank the learners.

5. RESULTS

The error values obtained from AR and MRE have been plotted and can be seen in Figure 3 and Figure 4. The summary we can conclude from this is that :

1. Non-analogy methods sometimes did better than the analogy-based estimators in some cases. Hence, analogy-based learners have a comparable performance with the non-analogy based methods

2. TEAK based learners are not worse when compared with the analogy based estimators.

While the principle behind TEAK was to remove the confusion present in the dataset which would lead to a better prediction, the results obtained did not conclude that. In some cases, TEAK was a better learner whereas in other cases, ABE offered a better predictor. Hence, we can conclude that there is no single learner which is optimal for all types of datasets.

6. REFERENCES

[1] B. W. Boehm. Software engineering economics. *Prentice Hall PTR*, 1981.

[2] C. Kemerer. An empirical validation of software cost estimation models. *Comm. of the ACM*, 30:416–429, 1987.

[3] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE Transactions on Software Engineering*, 38:425–438, 2011.

[4] E. Kocaguneli, T. Menzies, and J. W. Keung. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering*, 38(6):1403–1416, 2012.

[5] B. Nuseibeh. To be and not to be: On managing inconsistency in software development. In *Proceedings of the 8th International Workshop on Software Specification and Design*, page 164. IEEE Computer Society, 1996.

[6] Spareref.com. Nasa to shut down checkout launch control system. 2002. http://bit.ly/eiYxlf.