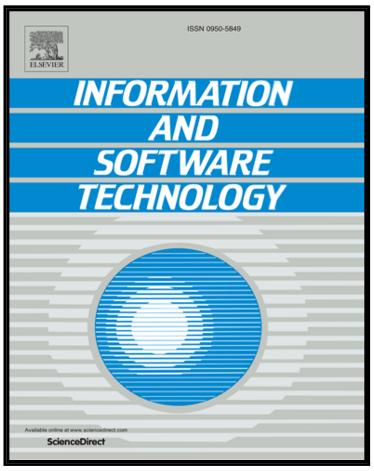


Software effort estimation based on open source projects: Case study of GitHub

Fumin Qi , Xiao-Yuan Jing , Xiaoke Zhu , Xiaoyuan Xie , Baowen Xu , Shi Ying

PII: S0950-5849(17)30223-9
DOI: 10.1016/j.infsof.2017.07.015
Reference: INFSOF 5858

To appear in: *Information and Software Technology*

Received date: 14 March 2017
Revised date: 5 July 2017
Accepted date: 31 July 2017

Software effort estimation based on open source projects: Case study of GitHub

Fumin Qi[a], Xiao-Yuan Jing[a,b,*], Xiaoke Zhu[a], Xiaoyuan Xie[a], Baowen Xu[a], Shi Ying[a]
[a]State Key Laboratory of Software Engineering, School of Computer, Wuhan University, China
[b]School of Automation, Nanjing University of Posts and Telecommunications, China
*Corresponding author: jingxy_2000@126.com

ABSTRACT

Context: Managers usually want to pre-estimate the effort of a new project for reasonably dividing their limited resources. In reality, it is common practice to train a prediction model based on effort datasets to predict the effort required by a project. Sufficient data is the basis for training a good estimator, yet most of the data owners are unwilling to share their closed source project (CSP) effort data due to the privacy concerns, which means that we can only obtain a small number of effort data. Effort estimator built on the limited data usually cannot satisfy the practical requirement.
Objective: We aim to provide a method which can be used to collect sufficient data for solving the problem of lack of training data when building an effort estimation model.
Method: We propose to mine GitHub to collect sufficient and diverse real-life effort data for effort estimation. Specifically, we first demonstrate the feasibility of our cost metrics (including functional point analysis and personnel factors). In particular, we design a quantitative method for evaluating the personnel metrics based on GitHub data. Then we design a samples incremental approach based on AdaBoost and Classification And Regression Tree (ABCART) to make the collected dataset owns dynamic expansion capability.
Results: Experimental results on the collected dataset show that: (1) the personnel factor is helpful for improving the performance of the effort estimation. (2) the proposed ABCART algorithm can increase the samples of the collected dataset online. (3) the estimators built on the collected data can achieve comparable performance with those of the estimators which built on existing effort datasets.
Conclusions: Effort estimation based on Open Source Project (OSP) is an effective way for getting the effort required by a new project, especially for the case of lacking training data.

Keywords: Software effort estimation; effort data collection; automated function point; open source project; AdaBoost.

1. INTRODUCTION

Software effort estimation is an important step for developing a software project, which has attracted many concerns of researchers [1-13]. Underestimated effort will lead to insufficient resources allocated to the project, and therefore result in the fact that the quality of the project cannot be guaranteed [14, 15]. On the other hand, overestimated effort generally implies high budget, such that the price of the software becomes unnecessarily high and lose its advantage in bidding [14]. Therefore, an accurate effort estimation is an essential part for developing a project [7, 16, 17].

Most of the effort estimation studies mainly focus on designing a good estimator [3, 18-28] or new cost metrics [16, 29-32] or other aspects [2, 14, 15, 33-41]. A systematic survey by Jørgensen and Shepperd [17] on 304 papers from 76 journals show that most of their surveyed papers focus on three topics (i.e., estimation methods, cost metrics and organizational issue), accounted for more than 70% of the total surveyed articles. For example, there exist many estimators have been developed, including expert-based method [19], analogy-based method [20] and model-based methods [3, 42-44]. Albrecht in [31] proposed a function point analysis method to evaluate the software project. Boehm [1] provided a COCOMO II cost model for software effort estimations. Researchers of [2, 14, 15, 33-41] have also made some other aspects of researches on effort estimations, including the selection of the estimators [41], investigating the influence of selection bias on effort overruns [14], and etc.

Existing effort estimation methods usually require enough data for training a good effort estimator. However, most of the data owners are unwilling to share their data [45] due to privacy concerns, which leads to a severe shortage of project data for real-life effort estimation.

1.1 Motivation

There are many methods try to indirectly alleviate the problem of data shortage. For example, Song et al. [18] proposed a new method which uses Grey Relational Analysis (GRA) of Grey System Theory (GST) for feature subset selection and effort estimation with small dataset. Menzies et al. [46] provided a platform for researchers to share their data. In [47], we have proposed a novel privacy-preserving method to solve the privacy problem in the process of effort data sharing. However, data shortage is not only in terms of the size of dataset. For example, in real-life effort estimation, a manager may suffer from the absence of project data that have similar business field or characteristics. Therefore, method in [47] can only alleviate one aspect of this problem. On the other hand, despite of the contributions in [18, 46], collecting data still requires that owners share their closed source project effort data. However, to the best of our knowledge, currently only a small number of organizations or institutions are willing to publish their owned/collected effort data

As mentioned above, currently only a small number of organizations or institutions are willing to provide service/publish their owned/collected effort data [46, 48], one large open dataset is International Software Benchmarking Standards Group (ISBSG) [48], which provides a charging version of effort dataset. Another one is Promise repository [46] opened for public as a free service funded by National Science Foundation (NSF), which provides various datasets to the users for academic research purposes. More specifically, the Promise repository consists of 13 effort datasets, which are collected by using 6 different costs metrics [3, 16, 29-31]. However, different industrial software projects have their own different characteristics, such as different business fields, different development platforms, and etc. Therefore, there is an urgent need of dataset sources which are large-scale, diverse, and similar to the real-life projects.

Nowadays, open-source project hosting platforms become more and more popular. One of the largest one, GitHub, is hosting over 53 million projects by the end of 2016[1]. These projects are diverse in terms of their types, domains, languages, and etc. More importantly, as an on-line version control system, GitHub records various data during the project development, such as each individual commit, contributor, issues and their fixing solutions, discussion, and etc. These data from all open-source projects on GitHub can be accessed via APIs, which are excellent data source for our effort estimation study. However, due to the large amount and the diversity of the raw data, extracting, processing and modeling the data for effort estimation can never be trivial tasks.

1.2 Contribution

In this paper, we propose a method to alleviate the data deficiency problem by adopting GitHub data. Specifically, the contributions of our method are four folds.

(1) We propose and implement a platform for GitHub data crawling and filtering, as well as extracting necessary information for effort estimation. As compared with any of the previous studies, our method can get sufficient and much more diverse effort data to train an estimator, such that the new projects are more likely to get accuracy effort estimations.

(2) We collectively adopt function point analysis and personnel metrics as the cost metrics in our estimation. In particular, we design a quantitative method for evaluating the personnel metrics based on GitHub data, which can provide smoother values to more objectively reflect the presonnel factor of a project.

(3) We propose ABCART algorithm to make the collected dataset have dynamic expansion capability. With the expansion of the dataset, which adaptability will be further enhanced.

(4) We perform large-scale experiments to verify the performance of our collected data[2]. Our collected dataset can be adopted as a benchmark for future studies.

[1] https://github.com/about

[2] All data can be accessed on:

https://sites.google.com/site/whuxyjfmq/home/see_osp_dataset

2. RELATED WORK

There exist some studies trying to alleviate the problem of training data deficiency [18, 41]. These methods can be divided into two groups: (1) methods that focus on estimators; (2) methods that focus on data expansion. The first group of methods aim to design effort estimators based on small dataset. For example, Song et al. [18] proposed a method of using grey relational analysis to select feature subset for training a predicting model based on small dataset. Menzies et al. [41] designed a COSEEKMO toolkit, which can choose the most appropriate estimator from the alternative estimators to estimate for small dataset. Although these methods can be used to train a predicting model with small datasets, the fact that the training dataset is small still. The second group of methods focus on how to expand existing datasets. For example, Menzies et al. [7] presented a WHERE+WHICH method which tries to get a combination of samples from many datasets and trains the effort models based on the obtained dataset. WHERE+WHICH can be used to solve the problem of small dataset at certain extent, yet this method was also limited by the number/size of public effort datasets. Peter et al. [45] designed the MORPH algorithm for protecting the privacy of defect prediction data sharing, then, in 2013, they proposed the enhanced MORPH+CLIFF [40] for improving the performance of privacy-preserving, these two methods are named as LACE1. In 2015, Peter et al. [49] proposed LACE2 which consists of LACE1 and a data sharing mechanism for solving the problem of defect prediction data sharing. In addition, there exist a few studies that focus on the privacy concerns of effort data sharing. For example, we [47] proposed an ICSD&MLBDO method to protect the privacy of effort data to be shared.

To better share knowledge, several organizations contribute their platform to public to store the research-related resources, such as Promise repository supported by National Science Foundation (NSF) [46], which provides many research resources, including the aspects of effort estimations, requirements evaluation, modeling, defect prediction, and etc. The effort datasets in this repository are collected by using six different cost metrics [16, 29, 42, 48]. For example, Menzies et al. [30] donated their NASA93 effort dataset, which was collected from NASA. Kitchenham dataset [16] are collected from a outsource company. However, these data are far from being sufficient for a real-life effort estimation.

Nowadays, Open Source Projects (OSPs) become more and more popular. We can get large quantities of OSPs from the Internet. Taking the website of GitHub as an example, we can not only manage ourselves projects, but also access the OSPs of other teams through the provided APIs. The source codes and active information of OSPs can be easily obtained from GitHub. However, there are some challenges to get effort data from GitHub, such as how to extract data and how to translate GitHub data into cost metrics, and etc.

3. OUR METHOD

3.1 Overview of our method

Our proposed method consists of three parts, which is illustrated as Figure 1.

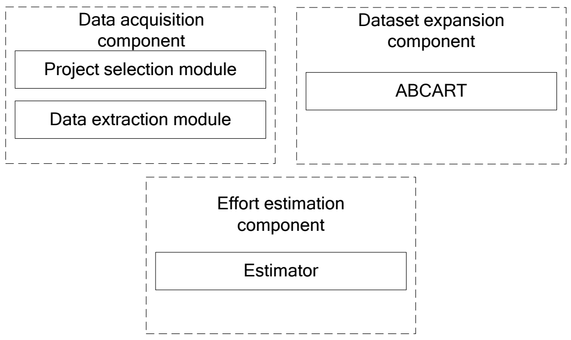

Data acquisition component
Project selection module
Data extraction module
Dataset expansion component
ABCART
Effort estimation component
Estimator

Figure 1 Overview of our method

Data acquisition component: This component is responsible for obtaining necessary data from OSP for effort estimation. The main functions of this component include scrapping projects from GitHub, filtering the collected projects, and extracting the necessary data from the filtered projects.

Dataset expansion component: To get more effort data to train the prediction models, we design this component to increase the samples of the collected dataset.

Effort estimation component: This component is responsible to train a prediction model on the collected data to estimate the effort of a new project.

In this paper, we mainly concentrate on the components of data acquisition and dataset expansion. For the effort estimation component, we use CART [43] as the estimator to predict the effort of a new project.

3.2 Data acquisition component

The data acquisition component consists of two modules: (1) project information selection module; (2) and effort data extraction module. The compositions of data acquisition component are shown in Figure 2.

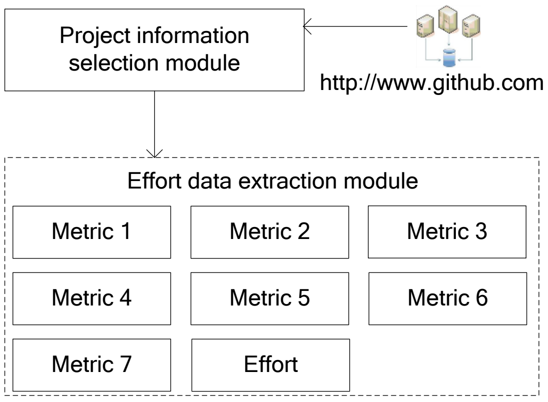


Figure 2 Compositions of data acquisition component

Project information selection module: This module is responsible to scrape relevant projects information from GitHub by using the user specified keywords, and filter the obtained raw projects information.

Effort data extraction module: This module is responsible for downloading and analyzing source codes of projects from GitHub, as well as extracting effort data of each project according to our cost model.

Next, we will individually introduce these modules.

3.2.1 Project selection module

There are thousands of open source projects stored on GitHub, where many APIs are provided for users to access project information. Assume that we have not enough projects as the reference object for estimating the effort required by a software system. Intuitively, we can obtain the reference projects from GitHub as the training data for constructing the prediction model. GitHub hosts over 53 million projects, it's impossible and unnecessary to analyze all the projects, therefore, how to select appropriate projects from these massive data is the first issue to be considered. We develop this module which takes the following three aspects into consideration to select project.

Development time: The projects we selected are from 2011 to 2016. With the progress of development language and IDE, as well as the increasing of user's requirements, the efforts required by developing a new software project may be different at different times. For example, we want to use a machine learning algorithm in a Java project (such as the support vector machine (SVM) [50] described by Vapnik et al. in 1995) to make a prediction in 2000, we may need to spend a lot of time to understand and rewrite this algorithm with Java language. This is quite time-consuming. However, in nowadays, with the available of many machine learning libraries, such as JAVA-ML [51] (which implements many commonly used machine learning algorithms and provides services through different interfaces), we can invoke the SVM algorithm in our program directly. Off-the-shelf libraries will improve the development efficiency of a project and decrease the effort required by a new project. Therefore, it is necessary to filter out a part of projects according to the developing time of the projects.

Team size: The team size of a project on GitHub can be evaluated by the number of contributors of this project. Since the project developed by a team with only one or two persons usually doesn't conform to the team configurations of most companies (e.g., Scrum guide recommends that development team size should be between 3 and 9 [52]), we require the team size of the project to be collected must be larger than 3.

Business-related: To train a good prediction model, sufficient training data is a required element, furthermore, it would be better if the collected effort data is similar to the projects to be estimated. As compared with the CSPs, we can easily get more business-related effort data of OSPs from GitHub. This requirement can be implemented by using the API of 'https://api.github.com/search/repositories?q=keywords'.

3.2.2 Data extraction module and cost metrics

The required effort for developing a project is determined by many cost metrics, such as LOC, FP, PLEX, and etc. [16, 31, 42]. We divide commonly used cost metrics into two groups: (1) function point analysis model, and (2) COCOMO II, which are summarized in Table 1.

Albrecht [31] published a dataset which is composed by 24 projects completed at IBM in the 1970s. Their projects are developed by using the third-generation languages, such as

COBOL, PL1, and etc. China dataset [46] consists of the software projects from different companies of China, which contains 499 samples with 17 cost metrics and one effort label. Kitchenham dataset [16] is composed by the effort data of different projects from a single outsourcing company, which contains 145 projects' effort data. Kemerer [32] published a dataset which is constructed by 15 software projects described by six cost metrics and one effort label. The Maxwell [3] dataset comes from the finance domain and is composed of Finnish banking software projects. It contains 62 projects described by 23 metrics. The Coc81 and Nasa93 datasets was collected by using the COCOMO II [53, 54] with the LOC metric.

Table 1 The cost metrics of some commonly used datasets

	Datasets	Cost metrics	
Function point analysis category	Albrecht	*Input, Output, Inquiry, File*, FPAdj, RawFP, *AdjFP*, Effort	
	China	*AdjFP, Input, Output*, Enquiry, *File*, Interface, Added, Changed, Deleted, PDR_AFP, PDR_UFP, NPDR_AFP, NPDU_UFP, Resource, Dev.Type, Duration, N_effort, Effort	
	kitchenham	Client.code, Project.type, Actual.start.date, Actual.duration, Actual.effort, *Adjusted.function points*, Estimated.completion.date, First estimate, First.estimate.method	
	kemerer	Language, Hardware, Duration, KLOC, *AdjFP*, RawFP, Effort	
	maxwell	Number of different development languages used (Nlan), Customer participation (T01), Development environment Adequacy (T02), Staff availability (T03), Standards use (T04), Methods use(T05), Tools use (T06), Software's logical complexity (T07), Requirements volatility (T09), Quality requirements (T10), Efficiency requirements (T11), Installation requirements (T12), Staff analysis skills (T12), Staff application knowledge (T13), Staff tool skills (T14), Staff team skills (T15), Duration, time, Application size(number of function points).	
COCOMO II category	Datasets	Cost metrics	
	Nasa93 Coc81	Product aspect	Required software reliability (RELY), Data base size (DATA), Product complexity (CPLX), Require reusability (RUSE), Documentation match to life-cycle needs (DOCU)
		Platform aspect	Execution time constraint (Time), Platform volatility (PVOL)
		Personnel aspect	Analyst capability (ACAP), Programmer capability (PCAP), Application experience (AEXP), Platform Experience (PEXP), Language and tool experience (LTEX), Personnel continuity (PCON)
		Project aspect	Use of software tools (TOOL), Multisite development (SITE), Required development schedule (SCED)

As compared with COCOMO II, it is easier for managers of a project to get the software size in the early project life-cycle by using Function Point analysis (FP) [31, 32]. Furthermore, many FP cost metrics are published as ISO standards, such as COSMIC [55], FiSMA [56], IFPUG [57], Mark-II [58] and NESAM [59]. Considering these two aspects, the FP is a suitable technique to be widely used for measuring the project. Furthermore, researchers usually use the Line Of Code (LOC) to measure the program size, without loss of generality, we add this metric into the cost model. As can be seen from Table 1, cost metrics of the datasets which based on function point analysis are not exactly the same with each other, yet, we also find that the cost metrics used by different datasets are partly overlapped. We select the common cost metrics (i.e., *italic style* words in Table 1) to construct our cost model. The rationales behind selecting the overlaped metrics of Table 1 as part of metrics of the cost model is that: Managers usually use cross company effort data to conduct effort estimation (i.e., Cross-company Software Effort Estimation, CSEE). The identical metric between different effort datasets is a basis requirement for commonly used CSEE predictors. In this paper, we also consider to create the basis (some identical metrics) for CSEE task when collecting effort dataset. Researches of [41, 60] show that personnel factor is one of important element for effort estimation, and some studies [60, 61] also verified this point. Therefore, we consider the personnel factor into our cost model (i.e., APEX and LTEX, we will introduce the two metrics in *B* section and give the reasons for choosing the two metrics). The cost metrics used in this paper are shown in Table 2.

Table 2 Cost metrics used in this paper

Cost metrics	Explanation
Input	Elementary process that processes data or control information sent from outside the boundary
Output	An elementary process that sends data or control information outside the boundary
Files	User recognizable group of logically related data or control information maintained within the boundary
AFP	Automated function point
LOC	Line of source code (including comments line)
APEX	The project team's equivalent level of experience in a particular type of applications
LTEX	The abilities of programming language and software tool experience of the project team for developing the software system or subsystem

A. Automated Function Point Measures for OSP

In this paper, we aim to collect the effort data from OSPs automatically. Therefore, we adopt the Automated Function Point (AFP) measures [62] into our study for measuring a software project, which is consistent with the function point Counting Practices Manual (CPM) maintained by the International Function Point Users' Group (IFPUG). We can use it to automatically analyze the project information and calculate the function point size of a project with few configurations. The procedure of AFP [62] is illustrated in Figure 3.

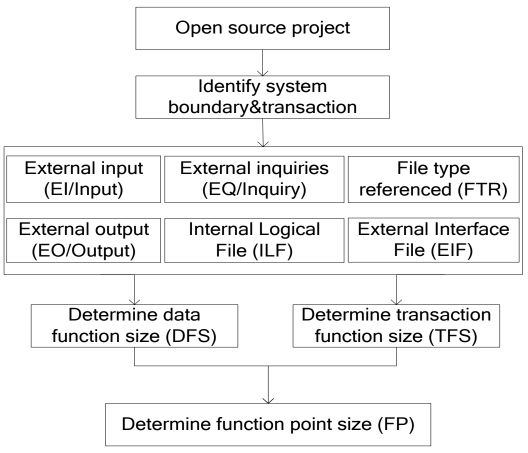

Open source project
Identify system boundary&transaction
External input (EI/Input)
External inquiries (EQ/Inquiry)
File type referenced (FTR)
External output (EO/Output)
Internal Logical File (ILF)
External Interface File (EIF)
Determine data function size (DFS)
Determine transaction function size (TFS)
Determine function point size (FP)

Figure 3 Flowchart of automated function point

In Figure 3, the identification of system boundaries refers to identifying the business scope boundaries of a project (i.e., some systems may consist of many subsystems), and its process requires the intervention of data collectors after parsing the source code of the project (We only need the users to tell the collector how many sub-systems the project may contain, default 1). In this paper, we take the Java project as an example. Java language uses the 'package' to solve the problem of naming collisions, which inspires us to use the naming convention to help identifying the system boundary.

For example, there are two computer systems in Wuhan University (WHU) developed by using Java language. One system is the campus Electronic CARD system (ECARD), and the other is the STUDENT management system (STUDENT). Their package names are provided as follows:

Package name 1: edu.whu.ecard.ui

Package name 2: edu.whu.student.dbop

It is easy for us to use their package names to identify the two systems through the naming convention. More detailed process of automated function point can be found in the automated function point version 1.0[3].

B. Designing a Quantitative Method to Evaluate the Personnel Metrics

COCOMO II considers the personnel metrics as one of the key factors for measuring a project [42]. The personnel metrics of COCOMO II consists of the following six aspects:

[3] http://www.omg.org/spec/AFP/

(1) Analyst CAPability (ACAP): Analysts are personals who work on requirements, high-level design and detailed design. ACAP focuses on the abilities of analysts, including analysis and design ability, efficiency and thoroughness, and the ability to communicate and cooperate with percentiles.

(2) Programmer CAPability (PCAP): It considers the rating of programmers related to the aspects of ability, efficiency and thoroughness, and the ability to communicate and cooperate.

(3) Personnel CONtinuity (PCON): It refers to the project's annual personnel turnover.

(4) Application experience (APEX): It refers to the project team's equivalent level of experience in a particular type of applications.

(5) PLatform EXperience (PLEX): It refers to recognizing the importance of understanding the use of more powerful platforms.

(6) Language and Tool EXperience (LTEX): It refers to the abilities of programming language and software tool experience of a project team for developing the software system or subsystem.

When predicting the effort required by a new project according to the COCOMO II operation manual [1], we can see that there exist no objective methods to evaluate some personnel metrics (such as ACAP, PCAP and PLEX), managers usually determine these personnel metrics by using their subjective judgment. The PCON is usually used to evaluate the project's annual personnel turnover. As compared with CSP, OSPs have no records that can be used to estimate their personnel mobilities. Therefore, PCONs of the OSPs are usually unpredictable, which lead to that PCONs lose their referential value to the actual effort estimations. For other personnel metrics (such as APEX and LTEX), managers can get them metrics by using the stored objective information (e.g., the number of a specific projects that have been developed by them). Considering the above aspects and the fact that the involved projects information of the contributors of the OSP is available to determine the APEX and LTEX, we choose APEX and LTEX as the personnel metrics of the cost metrics in OSP-based effort data.

For a personnel metrics, sometimes, the method that simply maps the input values to qualitative values cannot provide subtle and accurate discrimination. For instance, in COCOMO II [1], managers usually use some discrete values (i.e., very low, low, normal, high, very high and extra high) to determine the rating levels of personnel metrics, such as the rating levels of APEX of COCOMO II listed in Table 3. Take a team with 11 months of development experiences as an example, we can classify the APEX level of this team into 'L' according to Table 3. However, if another team has 12 months of development experiences, the APEX of this team can then be identified into category 'N'. From this example, we can see that teams may be assigned to different levels due to a small gap between the development experience months of these teams. Such categorization is not very practical in real-life cases. Therefore, a method with smoother distinguish method is needed to transform the qualitative values into the quantitative values, especially when we want to use some regression estimators for the effort estimation task. In this paper, we adopt the sigmoid function [63] to represent our personnel metrics (*PM*).

$$PM = \frac{1}{1+\exp(-\alpha(x-\beta))} \quad (1)$$

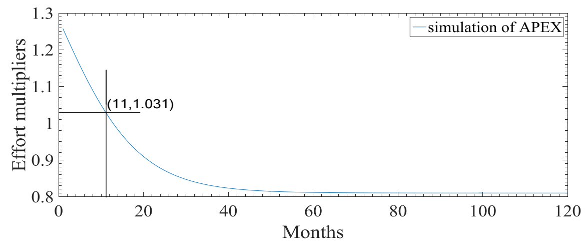
where α determines the range of translations of the curve, the β represents the ranges of width of the curve, x denotes the value of the personnel metric before transforming into a multiplier. Here, we use the APEX of COCOMO II as the example to illustrate the transformed APEX. In our paper, the *PM* value ranges from 0 to 1. For the APEX, the larger the value is, the higher the experience level in a particular type of applications of the team is. For the LTEX, the larger value is, the higher the programming language and software tool experience level of the team for developing the software system or subsystem is. The transformed APEX is shown in Figure 4. For demonstration purpose, we add the minimal multiplier value of the APEX onto the ordinate of the Figure 4.

Table 3 APEX of COCOMO II [1]

months	≤ 2	6	12	36	72	
levels	VL	L	N	H	VH	EH
multipliers	1.22	1.10	1.0	0.88	0.81	n/a

VL=very low, L=low, N=normal, H=high, VH= very high, EH= extra high..

In Figure 4, a team has 11 months ($x=11$) of experience in a particular type of applications, we can get 1.031 as the effort multiplier of their APEX, which is a smoother value. The determination of parameters of the formula (1) will be introduced in subsection 4.5.



take α =-0.105, β =-1 as illustrations

Figure 4 Sigmoidal function with specific parameters

For the LTEX, we also use this method with different parameters to calculate.

C. Counting the Effort made by an OSP Team

Before introducing the proposed effort counting method, we define three terms as follows.

Commit: Activities that developers submit their codes in a project. We use the commit data of a contributor and his active days to classify the type of this contributor (i.e., an active contributor or an inactive contributor).

Active day: Days that a contributor performs at least one commit to the repository.

Active-contributor: For an OSP, we call the contributor that have frequent contribution records as active contributor. The active contributor general make more contributions than other contributors. In contrast, for the contributors that contributions generally smaller than active contributors, we call them as inactive contributor. Specifically, for an OSP of GitHub, we can get the contributors and their committing information by API[4] and API[5] respectively. In [60], Robles et al. stated that if a contributor devotes over 12 commits per six months for a specific OSP, they can identify this contributor as full-time contributors with 75% confidence. Comparing to [60], we have the stricter criteria to identify the active contributor. In our method, we assume that: in a period of consecutive active days (default 30 days in our method), if a contributor has an average of one committing record per active day, and whose total contributions exceed the average contributions of the project, we call he/she as active contributor, otherwise we call he/she as inactive contributor. Therefore, we have the reason to believe that we can identify an active-contributor with above 75% confidence. For such a special case that a contributor ‘Alice’ is active for some period and then leaves the team, the contribution rate of Alice will be weakened by the total contributions of other contributors as the project progresses toward completion. In this case, Alice will not satisfy the criterion 2, and thus will not be identified as an active contributor.

In this section, we adopt the effort counting approach based on the fact that: most of the contributions of an OSP are made by active contributors. Robles et al. [60] presented a method to map the activity traces of OSPs to effort, which provides a way to count the effort of an developed OSP. We adopt their basic idea to count the effort made by an OSP team.

For the unit of effort required by a new project, we use the person-month (Measure of required effort for developing a software project [64-66]). For the OSP, contributors are composed by active contributors and inactive contributors, where different type of contributors have different number of commits records. Therefore, the calculations of person-month between OSPs and CSPs are different. Similar to the calculation of the person-month of [66], the person-month of OSP can be calculated by using formula (2):

$$person-month = ac*C_{ac}*k + iac*C_{iac}*k \quad (2)$$

where, ac denotes the number of active contributors, iac represents the number of inactive contributors, C_{ac} means the percentage of contributions provided by active contributors, and the $C_{iac}=1-C_{ac}$ represents the percentage of contributions devoted by inactive contributors. k denotes the months taken by a project from the beginning of the project to the completing of the project. Here, we give an example to illustrate the calculation of person-month required by an OSP:

Example 1: given an OSP completed by 5 contributors in 4 months, where 3 of 5 contributors are active contributors and remaining 2 peoples are inactive contributors. Here, we assume that the 3 of 5 contributors provide 90% contributions, while the remaining devote 10% contributions. Then, the person-months of this OSP can be calculated as follows:

$$person-month = 3*0.9*4 + 2*0.1*4 = 11.6$$

[4] https://api.github.com/repos/user/projectname/contributors

[5] https://api.github.com/repos/user/projectname/commits

3.3 Data expansion component

When training a prediction model, it is reasonable to choose data from projects with keywords that are strongly-relevant to current project. However, such strongly-relevant projects sometimes do not provide sufficient data for analysis. To solve this problem, we propose to include 'less-relevant' projects to complement the strongly-related data. Specifically, we start from the most relevant keywords (*keywords 1*), obtaining the corresponding project data; and then carefully choose *other keywords* to include new project data such that we always guarantee that the expanded data have better analysis results than previous data.

We call the obtained effort data by *keyword 1* as an *initial data*, and call the obtained effort data by *other keywords* as an *incremental data*. Thus, *how to combine the initial data and incremental data into a single dataset is a valuable research problem*. (We only prepare one new sample for each sample of initial dataset to construct the incremental dataset in experiment. Different sub-keywords can get different numbers of projects from GitHub for incremental dataset. The size of incremental dataset relies on two aspects, including 1) the users' requirement for the number of effort data, and 2) the number of projects that are related to sub-keywords projects hosted in GitHub. Users can prepare a larger size of incremental dataset to increment the initial dataset. Theoretically, the more samples contained in the incremental dataset, the higher probability the final dataset has to get more samples.)

The *initial data* are based on the *keyword 1*, therefore, there exists strong relationship between *initial data* and the project to be estimated. As compared with *initial data*, the *incremental data* are 'less-relevant' to the project to be estimated. Assume that we integrate the whole *incremental data* and the *initial data* into a final dataset directly, the performance of the predictors trained on the final dataset may be affected by the noisy samples of *incremental data*. Therefore, we need an effective strategy for avoiding the negative influence of *incremental data* to the final version of effort dataset. In the following, we describe the process of AdaBoost algorithm and our AdaBoost-based data incremental approach.

Given $D=\{(x_1,y_1),(x_2,y_2),\ldots,(x_i,y_i),\ldots,(x_N,y_N)\}$ denotes the *initial data* with N samples, where (x_i,y_i) represents the i^{th} instance of D, $x_i \in 1\times K$ denotes the samples with K cost metrics, and y_i denotes the effort/cost the i^{th} project. We use $\bar{L}(D)$ to represent the trained estimator/model on D. When we need to judge the state of a new sample (x_{N+1},y_{N+1}) that whether it can be added into D, we can firstly put this sample into D. Then we can get the new D, denoted by $D^{'}=\{(x_1,y_1),(x_2,y_2),\ldots,(x_i,y_i),\ldots,(x_N,y_N),(x_{N+1},y_{N+1})\}$. Next, we re-train a predictive model on $D^{'}$, denoted as $\bar{L}(D^{'})$. Finally, whether the sample (x_{N+1},y_{N+1}) can be merged into the *initial data* can be decided according to the performance between $\bar{L}(D^{'})$ and $\bar{L}(D)$: If the performance of $\bar{L}(D^{'})$ outperforms that of $\bar{L}(D)$, we add (x_{N+1},y_{N+1}) into D, otherwise (x_{N+1},y_{N+1}) is dropped.

From the previous analysis, we can see that a strong estimator is a necessary element in determining whether an incremental sample can be added into the initial dataset. However, there is no such strong predictor in effort estimations [67, 68] that can strongly assert that a new sample can be added into the initial dataset or not. Intuitively, we can employ the voting result of multiple predictors to solve this problem.

For example, lets $L(D)$ denotes an estimator set trained on D, $L(D)=\{\bar{L}_1(D),\bar{L}_2(D),\ldots,\bar{L}_j(D),\ldots,\bar{L}_T(D)\}$, where $\bar{L}_j(D)$ represents the j^{th} estimator trained on D. T denotes the number of estimators. For a new sample (x_{N+1},y_{N+1}), we firstly add it into D, get the temporary $D^{'}$ and re-train $\bar{L}_j(D^{'})$, and then we can use the result of formula (3) to determine whether this incremental sample can be added into D.

$$add=\begin{cases} yes,\ if\ \sum_{i=1}^{T} sign(\bar{L}_i(D^{'}))\geq 0 \\ no,\ otherwise \end{cases}$$
$$sign(\bar{L}_i(D^{'}))=\begin{cases} 1,\ if\ \bar{L}_i(D^{'})\ better\ than\ \bar{L}_i(D) \\ -1,\ otherwise \end{cases} \tag{3}$$

The formula (3) is a simple voting method for determining whether a new sample can be added into the initial dataset. The main limitations of formula (3) are listed as follows:

(1) The advantage of a single estimator is easily offseted by other estimators. For example, a strong estimator that is trained on the dataset $D^{'}$, outputs '1' ($sign(L_1(D^{'}))=1$), indicating that the incremental sample can be added into *initial data*, at the same time, another weak estimator, which is also trained on the dataset $D^{'}$, gives an opposite conclusion. Thus, the result of the strong estimator may be counteracted by other weak estimators.

(2) It is hard to eliminate the gap between different estimators. For example, one estimator is CART [43], the other estimator is RBFN [44] , and this may lead to the limitation (1) [6].

In fact, combining multiple weak predictors (i.e., base learners) to a strong predictor is an ensemble learning problem. Schapire [69] proposed the Boosting algorithm based on the question: '*Can a set of weak learners create a single strong learner?*'. The classification ability of a weak learner is slightly correlated with the true classification. For a strong learner, it is arbitrarily well-correlated with the true classification. Freund and Schapire [70] proposed an AdaBoost algorithm, which is one of the classic Boosting algorithms. Friedman [71] proposed a method for the estimation task, which uses the least-squares regression as the base learner of AdaBoost, known as the LSBoost method. LSBoost algorithm is usually used to solve the type of regression problems. However, LSBoost has its own inherent limitation, that is, under the influence of least-squares regression method, the estimated results may occur negative values, these values are obviously not allowed for software effort estimations. AdaBoost can change different base learners for creating different types of ensemble learners. CART is a commonly used estimator for software effort estimation [6] , which inspires us to design an ensemble learner by using CART as the base learner of AdaBoost. Then we can use the designed ensemble learner to determine whether a new sample can be added into the initial dataset. We name this strategy as AdaBoost based on CART estimator

(ABCART). Then, the formula (3) can be transformed into the formula (4):

$$add = \begin{cases} yes,\ ABCART(D^{'})\ better\ than\ ABCART(D) \\ no,\ otherwise \end{cases} \quad (4)$$

Before describing the ABCART algorithm, we firstly introduce some basic definitions that will be used in the following paragraphs.

Evaluation measure: We use Pred(25) [26, 30, 41] as a measure to evaluate the decision results. Pred(25) is defined as the percentage of estimated values falling within 25 percent of the actual values:

$$\Pr ed(25) = \frac{100}{N}\sum_{i=1}^{N}\begin{cases}1,\ if\ \ MRE_i \le 0.25 \\ 0,\ otherwise\end{cases}$$

where, $MRE_i = \frac{|y_i - \bar{y}_i|}{y_i}$ represents the magnitude of relative error (MRE) of x_i, $\bar{y}_i$ denotes the predicted effort of x_i, y_i represents the ground-truth of x_i.

The original AdaBoost algorithm is shown in **Algorithm 1** [72].

AdaBoost algorithm is usually used for the classification task (i.e., qualitative problem). However, the effort estimation is a quantitative problem, therefore, we make the following three aspects of changes to the original AdaBoost algorithm for the effort estimation task:

1） We set CART, which is a common used estimator in effort estimation, as the base learner of the AdaBoost.

2） In effort estimation, if the predicted effort values falling within 25 percent of the actual values, we regard it as a correct answer, otherwise a wrong. Therefore, we modify the error calculation step as $\varepsilon_t = \Pr ed(25)_t$.

3） We do not need to update the weight values, if the predicted effort of x_i falling within 25 percent of the actual value, otherwise, we update the weight values.

The process of proposed samples increment algorithm is shown in **Algorithm 2**.

Although the AdaBoost algorithm can combine multiple weak learners for creating a single strong learner. However, Adaboost algorithm has an inherent limitation: the performance of AdaBoost algorithm sensitive to the noisy samples [73, 74] (i.e., the samples who have significantly different data distribution from other data). To alleviate the negative effects on effort estimation performance brought by noisy data, we introduce the Interval Covering based Subclass Division (ICSD) method, which is presented in our previous paper [47] to remove the noisy samples before running the ABCART algorithm. ICSD is designed to convert the samples without class labels into labeled data (i.e., samples with class labels). ICSD can classify the noisy samples into a new subclass that owns just one single sample. After obtaining the classified data, we regard the subclasses that contain only one sample as the noisy data to be ruled out. Detailed steps of ICSD method is shown as follows:

Step 1: Get the tolerance error range of effort y_i of each sample (i.e., upper boundary and lower boundary of y_i, according to the Formula (5)). We use $YR = \{[y_1^l, y_1^u],...,[y_i^l, y_i^u],...,[y_n^l, y_n^u]\}$ to represent the obtained ranges of all samples

$$\begin{cases} y_i^l = y_i - \lambda * y_i \\ y_i^u = y_i + \lambda * y_i \end{cases} \quad (5)$$

where y_i^u represents the upper bound and the y_i^l denotes the lower bound.

Algorithm 1: Original AdaBoost algorithm

```
Input: $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), \ldots, (x_N, y_N)\}$ , base learner L(D), iterate number T.
Procedure:
1: $W_1(x) = 1/N$ .
2: for t=1, 2, …, T
    3: $h_t = L(D, W_t)$ ;//obtain the trained model
    4: $\varepsilon_t = P_{x\sim W_t}(h_t(x) \ne y_i)$ ;
    5: if $\varepsilon_t$ >0.5 then break
    6: $\alpha_t = \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t})$ ;
    7: $W_{t+1}(x) = \frac{W_t(x)}{Z_t} \times \begin{cases}\exp(-\alpha_t),\ if\ \ h_t(x) = y_i \\ \exp(\alpha_t),\ otherwise\end{cases} = \frac{W_t(x)\exp(-\alpha_t y_i h_t(x))}{Z_t}$
8: end for
Output: $sign(\sum_{t=1}^{T}\alpha_t h_t(x))$
```

where y denotes the ground-truth of x, α_t is the classifier weight factor; $W_i(x)$ is the weighted Matrix. Z denotes a normalization factor to make $W(x)$ always maintain a distribution.

Step 2: Obtain the coverage number of each y_i, (i.e., for each y_i, we calculate how many ranges in YR covers y_i by using Formula (6)). We use **C** $= \{C_1,...,C_i,...,C_n\}$ to denote the coverage numbers of all samples, where C_i is the coverage number of y_i.

$$C_i = \sum_{j=1}^{n} C_{ij} \quad (6),$$

where $C_{ij} = \begin{cases}1,\ if\ \ y_i \in [y_j^l, y_j^u] \\ 0,\ otherwise\end{cases}$.

Step 3: Sort the samples according to the C in ***ascending order***. For the i^{th} sample that has not been categorized into any subclass, we classify all the samples covered by the tolerance error range of y_i into a new subclass if these samples have not been labeled. The reason for using the ***ascending order*** to sort the samples is that: If a sample has higher covering number, more samples may be covered by the range of the sample with a higher possibility. Therefore, assume that we sort samples according to the descending order of C , and then the generated subclasses may suffer the class imbalance problem [75].

Step 4: For the subclasses with only one sample, we offer two ways to process it: (1) discarding the sample of it. In our opinion, this kind of sample can be regarded as noisy samples that may affect the effort estimation of a new project; (2) Classifying it into a nearest subclass.

Algorithm 2: ABCART algorithm

```
Input: $D=\{(x_1,y_1),(x_2,y_2),\ldots,(x_i,y_i),\ldots,(x_N,y_N)\}$ ,
base learner L(D), iterate number T (default 10).

Procedure:
1: $W_1(x)=1/N$ .
2: for t=1, 2, …, T
    3:  $h_t=L(D,W_t)$ ;//obtaining the trained model
    4:  $P_t=\begin{cases}1, if\ \frac{|(h_t(x)-y_i)|}{y_i}\le 0.25\ (i.e.\ MRE_i\le 0.25)\\ 0, otherwise\end{cases}$
    5:  $\varepsilon_t=\sum_{i=1}^{N}\frac{P_i}{N}$ ;
    6:  if $\varepsilon_t$ >0.5 then
            break; T=t;
        end if
    7:  $\alpha_t=\frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t})$ ;
    8:  $W_{t+1}(x)=\frac{W_t(x)}{Z_t}\times\begin{cases}\exp_t(-\alpha_t), if\ P_i=1\\ \exp(\alpha_t), otherwise\end{cases}$
                 $=\frac{W_t(x)\exp(-\alpha_t P_t)}{Z_t}$
9: end for

Output: $\frac{1}{\sum_{i=1}^{T}a_i}\sum_{t=1}^{T}a_t h_t(x)$
```

4. EXPERIMENT

4.1 Research questions

To systematically evaluate our OSP-based effort estimation method, we set three research questions.

RQ1: Can the designed personnel metrics help to the OSP-based effort estimation?

RQ2: Is ABCART a suitable way to increase the instance of the collected dataset?

RQ3: Whether the collected dataset can be adapted to multiple estimators?

RQ1, RQ2, and RQ3 lead us to investigate whether our method can be used to solve the problem of lack of training data for the training of estimators.

4.2 Measures

In this paper, we use three measures, namely Pred(25) and MdMRE to evaluate the experimental results, which are commonly used measures for evaluating the effort estimation performance of estimators. The Pred(25) denotes the percentage of estimated values falling within 25 percent of the actual values:

$$\mathrm{Pred}(25)=\frac{100}{N}\sum_{i=1}^{N}\begin{cases}1, if\ MRE_i\le 0.25\\ 0, otherwise\end{cases} \tag{7}$$

Given a sample x_i with the actual effort being y_i, if the predicted effort is $\bar{y}_i$, the Magnitude of Relative Error (MRE) of x_i can be calculated by $MRE_i=\frac{|y_i-\bar{y}_i|}{y_i}$. MdMRE [6, 7, 11] of the N samples can be computed as

$$MdMRE=median(MRE_1,\ldots,MRE_i,\ldots,MRE_N)\ .$$

The above two measures are based on the MRE. Here, we introduce the third measure no-MRE based measure to evaluate the experimental results, namely Relative Standard Deviation (RSD) proposed by Foss et al. [76]. The RSD is formulated as follow:

$$RSD=\sqrt{\frac{\sum(\frac{y_i-\bar{y}_i}{v_i})^2}{N-1}}\ ,$$

where, v_i represents the AFP metric of the i^{th} sapmle of test data. When using the RSD to evaluate the performance of COCOMO II-based dataset, v_i represents the LOC metric of the i^{th} sapmle of test data (because both AFP and LOC are associated with product factor. Sometimes, two matrics can be transformed from each other. Therefore, we select LOC as the v_i of RSD to evaluate the NASA93 and COC81 datasets).

For these three measures, the higher Pred (25) measure means the prediction model more precise, and the lower MdMRE and RSD measures represent the better performance of the predicting model.

4.3 Collected dataset

Different users/applications have their own business requirements. Here, we only give a case study of GitHub as an example to illustrate our open source project (OSP) based effort estimation. In this paper, all the effort data are extracted from the OSPs of GitHub.

Assume that we use 'Java' as the *keyword 1* to request top 1000 projects information from GitHub through the API. (GitHub only returns top 1000 projects information per specific keyword when the number of the relevant results is larger than 1000). In this work, we firstly get 91 projects from the raw data and extract the effort data of these projects by using the methods introduced in section 3.2. We store the extracted effort data as the *initial data*. Next, we change another keyword 'Java application' to request more projects from GitHub, and use the filtering module to select the projects and extract their effort data as *incremental data* (containing 91 samples). We further use the ABCART method to determine whether the samples of *incremental data* can be combined into the *initial data*set. Finally, we use the combined data as the collected effort dataset. In this paper, the collected effort dataset contains 147 samples.

We conduct an experiment on the combined dataset to investigate their statistical properties. In addition to reporting the basic characteristics of the collected dataset, we divide the dataset into 3 parts according to the quantiles of the effort values (i.e., 0 ~ 25%, 25 ~ 75% and 75%), and analyze these three parts. The statistical results are shown in Table 4.

Table 4 Overview of our collected dataset

Effort ranges	Number of samples	mean	median	Standard deviation (SD)
[0~Q1)	44	78.32	79.5	32.62
[Q1~Q3]	93	275.10	260	118.43
(Q3~)	45	1440.84	1021	1153.03
[0~]	182	515.76	260.5	787.15

Q1=128; Q3=544, [0~] represents the range of effort values of the whole data that *initial data* add the *incremental data* directly

4.4 The determination of parameters of the personnel metrics

In formula (1), different personnel metrics have different rating levels driven by various elements. For example, we use APEX to evaluate the application experiences of a team, and use LTEX to measure the language and tool experiences of a team. Thus, we need to seek different parameters α and β for formula (1) to represent those two different cost metrics.

To determine the parameters values used in APEX and LTEX, we conduct a statistical study on an initial personnel data of OSPs. We firstly get 210 personnel data of teams of OSPs by using keyword of 'Java' from GitHub, then use the data of active contributors to evaluate the APEX and LTEX of a team. The information of a contributors' project experience (such as number of developed projects, development language of each project, and etc.), can be scraped by using GitHub APIs. We firstly investigate the data distribution of 210 obtained teams, and draw the quartile box plots of the APEX and LPEX data, and results are shown in Figure 5.

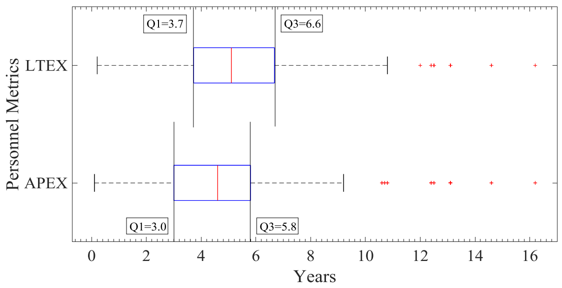
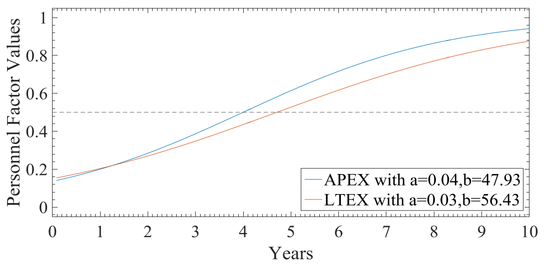


Figure 5 Personnel metrics of open source projects

From Figure 5, we can see that most of the teams of Java projects have 3.0 to 5.8 years of development experience and 3.7 to 6.6 years of Java language and software tool experience. We use the formula (1) to fit the data of Figure 5, the results are shown in Figure 6.

In Figure 6, the values of α and β for the APEX are 0.04 and 47.93 respectively, and the values of α and β for the LTEX are 0.03 and 56.43 respectively. After obtaining these parameters, we can easily get the APEX and LTEX of a team by using formula (1) with these obtained parameters values. The process of fitting the discrete values to continuous values can be implemented by using nonlinear regression fitting method (e.g., the nlinfit function provided by MATLAB version 2015b).



The y coordinate represents the empirical value

Figure 6 Fitting results of formula (1) on personnel data

4.5 Experimental design and result

For each dataset of the following experiments, we randomly select 50% samples as training data and the remaining as the test data. We construct the estimators on training data and use the trained models to predict the test data. To avoid the negative effect on effort estimations brought by the random selection strategy, we conduct experiment 20 times and report the average results.

4.5.1 *Experiment for personnel metrics **(RQ1)***

To verify whether the personnel metrics of OSP can improve the performance of effort estimations, we firstly generate a subset of the collected dataset (i.e., the subset consists of the cost metrics of input, output, FP, LOC, files and effort). Then, we use the CART (without pruning) as the baseline estimator, and train it on the original dataset and the generated subset, and use the trained models for testing. The Pred(25) and the MdMRE are used as the evaluation measures. The results of this experiment are shown in Table 5.

Table 5 Results of experiment on personnel metrics

	Pred(25)	MdMRE	RSD
X^c	37.73±5.81	0.36±0.06	0.25±0.06
X^{c^-}	26.09±0.08	12.52±2.4	0.38±0.08

X^c of Table 5 represents the original dataset, and X^{c^-} denotes the subset of collected dataset, which consists of the cost metrics except APEX and LTEX.

In this subsection, we firstly build the prediction models on the collected dataset (experiment p1) without considering personnel metrics (i.e., ATEX and LTEX metrics); Then, we train another predicting model on the complete collected dataset (experiment p2). As compared with the results of experiment p1, the experiment p2 has better performance. This result indicates that: even if the number of the functional point of two projects is similar, the efforts required by these two projects may be very different. One of the reasons is the personnel factor. The designed personnel metrics is a helpful element for our OSP-based effort estimation.

4.5.2 *Experiment for instance incremental strategy (**RQ2**)*

To check the effectiveness of ABCART algorithm, two types of data incremental experiments are conducted as follows:

Let X denote the *initial data*set and I represent the *incremental data*. In the first experiment, we use the ABCART to increase the samples of *initial data*set, and use the X^c to denote the obtained dataset; and in the second experiment, we use the $X^a = X + I$ to represent the dataset by directly combing the X and the whole *incremental data I*. We evaluate these experiments by using Pred(25) and MdMRE, and use the CART (without pruning) as the baseline estimator. The results of these experiments are reported in Table 6.

Table 6 Comparison of different incremental strategies

	Pred(25)	MdMRE	RSD
X	28.56±6.42	0.44±0.07	0.32±0.08
X^a	31.92±7.45	0.39±0.06	0.43±0.07
X^c	36.64±5.88	0.36±0.04	0.25±0.04

We can increase the samples of effort dataset by using the incremental data. Theoretically, we should not directly combine the whole of the *incremental data* into the *initial data* directly, since the correlation between sub-relevant keywords and the project to be developed is weaker than that of the first keywords. Therefore, how to increase the samples of initial dataset online is a valuable research problem. Firstly, we merge *incremental data I* and *initial data* X into a single dataset (i.e., X^c) by using ABCART algorithm (strategy 1), and use the obtained dataset for modeling and testing. Secondly, we merge *incremental data I* and *initial data* X as a single dataset (i.e., X^a) directly (strategy 2), and train the predicting model on the dataset X^a for testing. From Table 6, we can see that the performance of strategy 1 is better than that of strategy 2. The reasons for this phenomenon can be summarized as the following two aspects:

(1) ABCART algorithm filters the noisy samples in *incremental data* and combines the remaining into the *initial data*. Then, we can get more samples for training the predicting model. ABCART improves the predicting capability of prediction models, meanwhile reduces the negative impact of *incremental data*;

(2) ABCART combines multiple weak learners to a stronger learner, which makes the judgement on incremental samples more accurate. As well as, ABCART provides a suitable way to increase the instance of collected dataset.

4.5.3 *Experiment for adaptive ability (**RQ3**)*

To validate whether the collected dataset can be used to build different estimators, we conduct the following experiments.

We select several commonly used estimators to verify the adaptive ability of the collected dataset, including classification and regression tree (CART) (with pruning) [43] partial least squares regression (PLS) [3], robust regression (RoR) [29], radial basis function neural network (RBFN) [44] and K-nearest neighbors (KNN) (where k=1) [18]. In addition, we also employ different estimators to build prediction models on commonly used datasets of effort estimation, and make a comparison on these datasets. We utilize the Pred(25) and MdMRE to measure the performance of estimators, and compare the adaptive ability by using the correlation coefficient (Coff) of Pred(25) and MdMRE between the estimators results of our collected dataset and those of compared CSP datasets. Let *A* and *B* denote two variables with *N* scalar observations, then the calculation of correlation coefficient is defined as formula (8).

$$\rho(A,B) = \frac{1}{N-1}\sum_{i=1}^{N}\left(\frac{A_i - \bar{\mu}_A}{\sigma_A}\right)\left(\frac{B_i - \bar{\mu}_B}{\sigma_B}\right) \tag{8}$$

where, μ_A and σ_A are the mean and standard deviation of *A*, respectively, μ_B and σ_B are the mean and standard deviation of *B*. The value of $\rho(A,B)$ ranges from -1 to 1, where −1 represents totally negative linear correlation, 0 represents no linear correlation, and 1 represents totally positive linear correlation. Related experimental results are shown in Tables 7~12.

Table 7 Results of different estimators with two measures on our dataset

	Pred(25)	MdMRE	RSD
CART	57.18±4.43	0.35±0.04	0.26±0.05
PLS	41.45±6.82	0.83±0.07	0.29±0.07
ROR	42.37±5.71	0.51±0.04	0.27±0.06
RBFN	31.00±6.12	1.44±0.06	0.31±0.09
KNN	30.91±6.69	1.60±0.09	0.33±0.08

Table 8 Results of different estimators with two measures on coc81 dataset

	Pred(25)	MdMRE	RSD
CART	15.32±4.75	0.88±0.38	4.73.56±1.25
PLS	6.65±3.04	1.63±0.61	8.78±2.56
ROR	8.39±2.84	1.02±0.76	6.73±1.83
RBFN	14.19±4.49	2.88±0.56	5.52±2.34
KNN	2.00±0.04	2.30±0.20	12.93±0.48
Coff	0.504	0.94	0.73

Table 9 Results of different estimators with two measures on nasa93 dataset

	Pred(25)	MdMRE	RSD
CART	35.04±5.27	0.42±0.06	0.45±0.06
PLS	14.89±6.59	1.24±0.61	1.96±0.66
ROR	24.65±7.00	0.62±0.16	8.56±2.35
RBFN	22.72±5.66	0.63±0.11	4.06±1.25
KNN	2.83±1.59	3.79±0.34	13.47±3.24
Coff	0.76	0.68	0.62

In experiments, the adaptive ability of different datasets is compared with the correlation coefficients of Pred(25) and MdMRE between our collected dataset and other CSP datasets (i.e., COC81, NASA93, china, kemerer and kitchenham). As can be seen from Table 7, the effort data collected from OSPs can be applied to several estimators. Tables 7~12 show that the effort data collected from OSPs' have comparable adaptive ability (the values of Coff are larger than 0.5) with those of other compared CSP datasets.

Table 10 Results of different estimators with two measures on china dataset

	Pred(25)	MdMRE	RSD
CART	49.68±3.56	0.25±0.02	1.69±0.25
PLS	14.38±6.16	1.38±0.94	11.36±2.34
ROR	35.34±3.09	0.37±0.03	4.66±1.62
RBFN	24.50±2.63	2.48±0.02	6.14±0.72
KNN	11.00±0.80	1.80±0.10	12.52±0.83
Coff	0.82	0.73	0.78

Table 11 Results of different estimators with two measures on kemerer dataset

	Pred(25)	MdMRE	RSD
CART	24.29±6.12	1.10±0.81	0.86±0.13
PLS	5.71±5.55	2.45±1.87	1.54±0.45
ROR	15.37±6.58	2.06±2.80	2.54±0.76
RBFN	14.34±7.01	3.41±0.28	3.82±1.12
KNN	3.02±0.05	3.00±0.50	8.04±2.34
Coff	0.74	0.91	0.88

Table 12 Results of different estimators with two measures on kitchenham dataset

	Pred(25)	MdMRE	RSD
CART	49.03±5.43	0.26±0.03	3.75±0.78
PLS	50.06±7.62	0.35±0.05	2.18±1.79
ROR	61.67±6.44	0.18±0.03	1.99±0.99
RBFN	29.03±4.87	0.46±0.06	5.30±0.86
KNN	9.01±5.03	1.30±0.10	10.96±0.61
Coff	0.68	0.80	0.82

5. FURTHER DISCUSSION&ANALYSIS

We sort the samples by the number of contributors in ascending order and investigate the distribution relationships between the active contributors and the inactive contributors. The distribution information about the contributors of the collected effort data is shown in Figure 7.

From Figure 7, we can see that: in most cases, the active contributors account for only a small portion of all contributors (from 10% to 30%). Our results are similar to those reported by Robles in 2014, however, the difference is that the proportion of our active contributors falls into wider ranges. The main reason for this phenomenon is that: the number and the size of our selected projects are different to those of Robles.

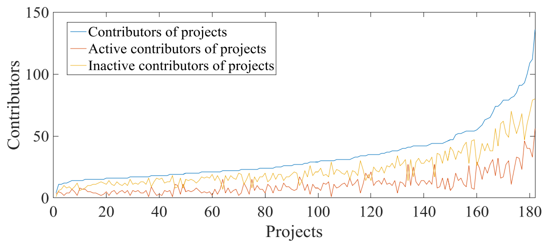


Figure 7 Relationship between active contributors and inactive contributors.

To investigate the relationships between the function points and the lines of code of the collected effort dataset, we performed an investigation experiment. The related results are shown in Figure 8.

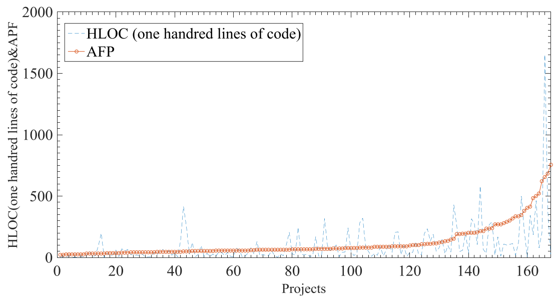


Figure 8 Relationship between function points and lines of code on the collected effort dataset

From Figure 8, we can see that there exists a positive correlation between LOC and AFP. However, there exists no obvious transformation relationship between the line of code and the function point. The reason for this phenomenon may be that: the effort data are collected from different projects and developed by different 'teams', where different 'teams' have different levels of development experiences.

To further simulate the real-life effort estimation (in real-life, the test dataset may contains more noisy sample), we use another measure to conduct the operation of sample incrementation and use the incremented dataset to conduct experiment. Specifically, we relax the measure that is used to evaluate the decision results, namely we use Pred(30) (like formula (7), Pred(30) denotes the percentage of estimated values falling within 30 percent of the actual values) as the threshold to evaluate that whether the sample of *incremental data* can be add into the *initial data*set. Here, we use the OSPv2 to denotes the obtained dataset. For experimental settings, we randomly select 50% samples as training data and the remaining as the test data. We use the CART (with pruning) as the baseline learner and use the Pred(25), and MdMRE, and RSD as the measures. We conduct experiment 20 times and report the average results. The results of the are reported in Table 13.

From the result of Table 13, we can see that after adding some noisy samples into the final dataset, the obtained dataset still can be adapting multiple estimators.

Table 13 Results of different estimators with two measures on OSPv2 dataset

	Pred(25)	MdMRE	RSD
CART	52.36±9.67	0.51±0.08	0.19±0.03
PLS	39.12±9.34	0.40±0.05	0.20±0.02
ROR	38.51±8.13	0.26±0.07	0.22±0.03
RBFN	28.28±6.32	1.51±0.11	0.30±0.25
KNN	27.29±6.91	1.74±0.12	0.35±0.04

6. THREATS AND VALIDITY

This section discusses the addressed and unaddressed threats to validity. It is organized around the three common types of validity threats.

6.1 Internal Validity

The MdMRE and Pred(25) and RSD measures used for reporting estimation results is a bias. Other measures, including Mean Balanced Relative Error (MBRE) [6], Mean Inverted Balanced Relative Error (MIBRE) [6] , CLUSTER [77] are not used. In this work, we employ the widely used MdMRE and Pred(25) and RSD measures to show the empirical evaluation for the application of software effort estimation.

In this paper, we only use 'Java program' related kewords to collect effort dataset. The studies of using more keywords to collect effort datasets should be done.

6.2 Construct Validity

The projects stored in the repository, which are associated with specific keywords may be few. We attempted to mitigate this issue by changing other diverse keywords. Therefore, the performance of the collected data may be affected by the keywords of user provided.

The factors used to filter the projects. In this paper, we only use three factors as the criterias to select project (i.e., dev.time, team size and bussiness-related). More factors may also can be used as the project filtering factors, such as the version of development language, development platform, and etc.

The criteria used to identify the active contributor. In this paper, we use two criteria to identify the active contributor. More studies on the identification of active contributor should be done.

6.3 External Validity

This article only uses the java open source projects of GitHub as a case study, since there exist many open source projects on GitHub server and it provides a complete version control, we can easily use the API to access more open source project information. Datasets collected by using other keywords or from other code hosting sites, such as BitBucket, may be needed for further analysis.

The hobbyist projects contained in the GitHub. In this paper, the impact of hobbyist projects for the final dataset may need be further studied.

The protection of GitHub for private repositories. Protection strategy in GitHub for private repositories that lets some commit informations of small part of personnels can not be obtained by data collector. More studies may need to be conducted on the team experiences.

7. CONCLUSION AND FUTURE WORK

In this paper, we propose an open source project based effort estimation method and provide a case study of GitHub. We not only employ the function point related measures as the cost metrics, but also take the personnel metrics into consideration for enriching the cost model.

To meet the needs of the dynamic growth of datasets, we developed an ensemble learning based sample filtering strategy (i.e., ABCART). For a new sample, we employ ABCART approach to check whether this sample can be added into the collected dataset. We conducted extensive empirical experiments on our collected dataset, including relationship experiment between different cost metrics, effectiveness experiment of dataset extension strategy and adaptability experiment of the collected data. The results show that the effort data collected from OSPs have comparable performance with most existed CSP effort datasets.

For the future work, we will provide more specific keywords related effort datasets as benchmark datasets from GitHub.

ACKNOWLEDGEMENTS

The authors want to thank the anonymous reviewers for their constructive comments and suggestions. The work described in this paper was supported by the National Nature Science Foundation of China under Projects No. 61272273, No. 61572375, No. 61233011, No. 91418202, No. 61472178, the Chinese 973 Program under Project No. 2014CB340702.

REFERENCES

[1] B. Boehm, C. Abts, B. Clark, S. Devnani-Chulani, COCOMO II model definition manual, The University of Southern California, (1997).
[2] M. Vangenuchten, Why Is Software Late - an Empirical-Study of Reasons for Delay in Software-Development, IEEE Transactions on Software Engineering, 17 (1991) 582-590.
[3] P. Sentas, L. Angelis, I. Stamelos, G. Bleris, Software productivity and effort prediction with ordinal regression, Information and Software Technology, 47 (2005) 17-29.
[4] B.A. Kitchenham, E. Mendes, G.H. Travassos, Cross versus within-company cost estimation studies: A systematic review, IEEE Transactions on Software Engineering, 33 (2007) 316-329.
[5] K. Dejaeger, W. Verbeke, D. Martens, B. Baesens, Data Mining Techniques for Software Effort Estimation: A Comparative Study, IEEE Transactions on Software Engineering, 38 (2012) 375-397.
[6] E. Kocaguneli, T. Menzies, J.W. Keung, On the Value of Ensemble Effort Estimation, IEEE Transactions on Software Engineering, 38 (2012) 1403-1416.
[7] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, T. Zimmermann, Local versus Global Lessons for Defect Prediction and Effort Estimation, IEEE Transactions on Software Engineering, 39 (2013) 822-834.
[8] Y.-S. Seo, D.-H. Bae, R. Jeffery, AREION: Software effort estimation based on multiple regressions with adaptive recursive data partitioning, Information and Software Technology, 55 (2013) 1710-1725.
[9] L.L. Minku, X. Yao, How to Make Best Use of Cross-Company Data in Software Effort Estimation?, in: 36th International Conference on Software Engineering (ICSE 2014), 2014, pp. 446-456.
[10] D. Čeke, B. Milašinović, Early effort estimation in web application development, Journal of Systems and Software, 103 (2015) 219-237.
[11] X.-Y. Jing, F. Qi, F. Wu, B. Xu, Missing data imputation based on low-rank recovery and semi-supervised regression for software effort estimation, in: 2016 38th International Conference on Software Engineering (ICSE), 2016, pp. 607-618.
[12] E. Løhre, M. Jørgensen, Numerical anchors and their strong effects on software development effort estimates, Journal of Systems and Software, 116 (2016) 49-56.
[13] G. Mathew, T. Menzies, J. Hihn, Impacts of Bad ESP (Early Size Predictions) on Software Effort Estimation, arXiv preprint arXiv:1612.03240, (2016).
[14] M. Jørgensen, The influence of selection bias on effort overruns in software development projects, Information and Software Technology, 55 (2013) 1640-1650.
[15] S. Grimstad, M. Jørgensen, K. Moløkken-Østvold, Software effort estimation terminology: The tower of Babel, Information and Software Technology, 48 (2006) 302-310.

[16] B. Kitchenham, S. Lawrence Pfleeger, B. McColl, S. Eagan, An empirical study of maintenance and development estimation accuracy, Journal of Systems and Software, 64 (2002) 57-77.
[17] M. Jørgensen, M. Shepperd, A systematic review of software development cost estimation studies, IEEE Transactions on Software Engineering, 33 (2007) 33-53.
[18] Q.B. Song, M. Shepperd, C. Mair, Using Grey Relational Analysis to predict software effort with small data sets, in: 2005 11th International Symposium on Software Metrics (METRICS), 2005, pp. 318-327.
[19] M. Jørgensen, Practical guidelines for expert-judgement-based software effort estimation, IEEE Software, 22 (2005) 57-63.
[20] A. Idri, F.a. Amazal, A. Abran, Analogy-based software development effort estimation: A systematic mapping and review, Information and Software Technology, 58 (2015) 206-230.
[21] E. Kocaguneli, T. Menzies, A.B. Bener, J.W. Keung, Exploiting the Essential Assumptions of Analogy-Based Effort Estimation, IEEE Transactions on Software Engineering, 38 (2012) 425-438.
[22] M.A. Ahmed, I. Ahmad, J.S. AlGhamdi, Probabilistic size proxy for software effort prediction: A framework, Information and Software Technology, 55 (2013) 241-251.
[23] E. Kocaguneli, T. Menzies, J. Keung, D. Cok, R. Madachy, Active Learning and Effort Estimation: Finding the Essential Content of Software Effort Estimation Data, IEEE Transactions on Software Engineering, 39 (2013) 1040-1053.
[24] L.L. Minku, X. Yao, Ensembles and locality: Insight on improving software effort estimation, Information and Software Technology, 55 (2013) 1512-1528.
[25] P. Chatzipetrou, E. Papatheocharous, L. Angelis, A.S. Andreou, A multivariate statistical framework for the analysis of software effort phase distribution, Information and Software Technology, 59 (2015) 149-169.
[26] P.A. Whigham, C.A. Owen, S.G. Macdonell, A Baseline Model for Software Effort Estimation, ACM Transactions on Software Engineering and Methodology, 24 (2015).
[27] B.K. Singh, A. Punhani, A.K. Misra, Integrated Approach of Software Project Size Estimation, International Journal of Software Engineering and Its Applications, 10 (2016) 45-64.
[28] J. Moeyersoms, E. Junqué de Fortuny, K. Dejaeger, B. Baesens, D. Martens, Comprehensible software fault and effort prediction: A data mining approach, Journal of Systems and Software, 100 (2015) 80-90.
[29] Y. Miyazaki, M. Terakado, K. Ozaki, H. Nozaki, Robust Regression for Developing Software Estimation Models, Journal of Systems and Software, 27 (1994) 3-16.
[30] T. Menzies, D. Port, Z.G. Chen, J. Hihn, S. Stukes, Validation methods for calibrating software effort models, in: ICSE 05: 27th International Conference on Software Engineering, Proceedings, 2005, pp. 587-595.
[31] A.J. Albrecht, J.E. Gaffney, Software Function, Source Lines of Code, and Development Effort Prediction - a Software Science Validation, IEEE Transactions on Software Engineering, 9 (1983) 639-648.
[32] S.R. Chidamber, C.F. Kemerer, A Metrics Suite for Object-Oriented Design, IEEE Transactions on Software Engineering, 20 (1994) 476-493.
[33] M. Jørgensen, D.I.K. Sjoberg, Impact of effort estimates on software project work, Information and Software Technology, 43 (2001) 939-948.
[34] M. Jørgensen, S. Grimstad, The Impact of Irrelevant and Misleading Information on Software Development Effort Estimates: A Randomized Controlled Field Experiment, IEEE Transactions on Software Engineering, 37 (2011) 695-707.
[35] C. Lokan, E. Mendes, Investigating the use of duration-based moving windows to improve software effort prediction: A replicated study, Information and Software Technology, 56 (2014) 1063-1075.
[36] E. Shihab, Y. Kamei, B. Adams, A.E. Hassan, Is lines of code a good measure of effort in effort-aware models?, Information and Software Technology, 55 (2013) 1981-1993.
[37] M. Jørgensen, K. Molokken-Ostvold, Reasons for software effort estimation error: Impact of respondent role, information collection approach, and data analysis method, IEEE Transactions on Software Engineering, 30 (2004) 993-1007.
[38] C.C. Wallshein, A.G. Loerch, Software cost estimating for CMMI Level 5 developers, Journal of Systems and Software, 105 (2015) 72-78.
[39] M. Jørgensen, Unit effects in software project effort estimation: Work-hours gives lower effort estimates than workdays, Journal of Systems and Software, 117 (2016) 274-281.
[40] F. Peters, T. Menzies, L. Gong, H.Y. Zhang, Balancing Privacy and Utility in Cross-Company Defect Prediction, IEEE Transactions on Software Engineering, 39 (2013) 1054-1068.
[41] T. Menzies, Z.H. Chen, J. Hihn, K. Lum, Selecting best practices for effort estimation, IEEE Transactions on Software Engineering, 32 (2006) 883-895.
[42] P.K. Oriogun, A survey of Boehm's work on the spiral models and COCOMO II - Towards software development process quality improvement, Software Quality Journal, 8 (1999) 53-62.
[43] J. Praagman, Classification and Regression Trees - Breiman,L, Friedman,Jh, Olshen,Ra, Stone,Cj, European Journal of Operational Research, 19 (1985) 144-144.
[44] V.S. Dave, K. Dutta, Neural network based models for software effort estimation: a review, Artificial Intelligence Review, 42 (2014) 295-307.
[45] F. Peters, T. Menzies, Privacy and Utility for Defect Prediction: Experiments with MORPH, in: 2012 34th International Conference on Software Engineering (ICSE), 2012, pp. 189-199.
[46] T. Menzies, Promise Repository, in, online, 2012.
[47] F. Qi, X.-Y. Jing, X. Zhu, F. Wu, L. Cheng, Privacy preserving via interval covering based subclass division and manifold learning based bi-directional obfuscation for effort estimation, in: 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), 2016, pp. 75-86.
[48] ISBSG, International Software Benchmarking Standards Group, in, 2011.
[49] F. Peters, T. Menzies, L. Layman, LACE2: Better Privacy-Preserving Data Sharing for Cross Project Defect Prediction, in: 2015 IEEE/ACM 37th International Conference on Software Engineering (ICSE 2015), 2015, pp. 801-811.
[50] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning, 20 (1995) 273-297.
[51] T. Abeel, Java Machine Learning Library, in, 2009.
[52] S. forum, Development Team Size, in, Scrum.org, 2017, pp. Development Team Size.
[53] G. Boetticher, T. Menzies, T. Ostrand, PROMISE Repository of empirical software engineering data, West Virginia University, Department of Computer Science, (2007).
[54] D. Reifer, B. Boehm, S. Chulani, The rosetta stone: Making cocomo 81 estimates work with cocomo ii, Crosstalk, (1999) 11-15.
[55] ISO/IEC19761:2011, Software engineering--COSMIC: a functional size measurement method, in, 2011.
[56] ISO/IEC29881:2010, Information technology--Systems and software engineering--FiSMA 1.1 functional size measurement method, in, 2010.
[57] ISO/IEC20926:2009, Software and systems engineering--Software measurement--IFPUG functional size measurement method 2009, in, 2009.
[58] ISO/IEC20968:2002, Software engineering--Mk II Function Point Analysis--Counting Practices Manual, in, 2002.
[59] ISO/IEC24570:2005, Software engineering--NESMA functional size measurement method version 2.1--Definitions and counting guidelines for the application of Function Point Analysis, in, 2005.
[60] G. Robles, J.M. González-Barahona, C. Cervigón, A. Capiluppi, D. Izquierdo-Cortázar, Estimating development effort in Free/Open source software projects by mining software repositories: a case study of OpenStack, in: 2014 ACM Proceedings of the 11th Working Conference on Mining Software Repositories, 2014, pp. 222-231.
[61] X. Wu, H. Wang, C. Liu, Y. Jia, Cross-view action recognition over heterogeneous feature spaces, in: Proceedings of the IEEE International Conference on Computer Vision, 2013, pp. 609-616.
[62] OMG, Automated Function Points, in, Object Management Group, online, 2014.
[63] M.N. Gibbs, D.J.C. MacKay, Variational Gaussian process classifiers, IEEE Transactions on Neural Networks, 11 (2000) 1458-1464.
[64] C. Sauer, A. Gemino, B.H. Reich, The impact of size and volatility - On IT project performance, Communications of the ACM, 50 (2007) 79-84.
[65] M. Alavi, An Assessment of the Prototyping Approach to Information-Systems Development, Communications of the ACM, 27 (1984) 556-561.
[66] U.o.N. Carolina, Calculating person months, in, https://www.engr.ncsu.edu/ora/preaward/documents/PersonMonths.pdf.
[67] A. Idri, M. Hosni, A. Abran, Systematic literature review of ensemble effort estimation, Journal of Systems and Software, 118 (2016) 151-175.
[68] J. Wen, S. Li, Z. Lin, Y. Hu, C. Huang, Systematic literature review of machine learning based software development effort estimation models, Information and Software Technology, 54 (2012) 41-59.
[69] R.E. Schapire, The boosting approach to machine learning: An overview, Nonlinear Estimation and Classification, 171 (2003) 149-171.
[70] R.E. Schapire, The Strength of Weak Learnability, Machine Learning, 5 (1990) 197-227.
[71] J.H. Friedman, Greedy function approximation: A gradient boosting machine, Annals of Statistics, 29 (2001) 1189-1232.
[72] Z.-H. Zhou, Ensemble learning, Encyclopedia of biometrics, (2015) 411-416.
[73] G. Ratsch, T. Onoda, K.R. Muller, Soft margins for AdaBoost, Machine Learning, 42 (2001) 287-320.
[74] H.X. Tian, Z.Z. Mao, An Ensemble ELM Based on Modified AdaBoost.RT Algorithm for Predicting the Temperature of Molten Steel in Ladle Furnace, IEEE Transactions on Automation Science and Engineering, 7 (2010) 73-80.
[75] H.B. He, E.A. Garcia, Learning from Imbalanced Data, IEEE Transactions on Knowledge and Data Engineering, 21 (2009) 1263-1284.
[76] T. Foss, E. Stensrud, B. Kitchenham, I. Myrtveit, A simulation study of the model evaluation criterion MMRE, IEEE Transactions on Software Engineering, 29 (2003) 985-995.
[77] J. Keung, E. Kocaguneli, T. Menzies, Finding conclusion stability for selecting the best effort predictor in software effort estimation, Automated Software Engineering, 20 (2013) 543-567.