Empirical Software Engineering (2019) 24:3153–3204
https://doi.org/10.1007/s10664-019-09686-w

A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation

Leandro L. Minku[1]

Published online: 26 February 2019
© The Author(s) 2019

Abstract
Software effort estimation is an online supervised learning problem, where new training projects may become available over time. In this scenario, the Cross-Company (CC) approach Dycom can drastically reduce the number of Within-Company (WC) projects needed for training, saving their collection cost. However, Dycom requires CC projects to be split into subsets. Both the number and composition of such subsets can affect Dycom's predictive performance. Even though clustering methods could be used to automatically create CC subsets, there are no procedures for automatically tuning the number of clusters over time in online supervised scenarios. This paper proposes the first procedure for that. An investigation of Dycom using six clustering methods and three automated tuning procedures is performed, to check whether clustering with automated tuning can create well performing CC splits. A case study with the ISBSG Repository shows that the proposed tuning procedure in combination with a simple threshold-based clustering method is the most successful in enabling Dycom to drastically reduce (by a factor of 10) the number of required WC training projects, while maintaining (or even improving) predictive performance in comparison with a corresponding WC model. A detailed analysis is provided to understand the conditions under which this approach does or does not work well. Overall, the proposed online supervised tuning procedure was generally successful in enabling a very simple threshold-based clustering approach to obtain the most competitive Dycom results. This demonstrates the value of automatically tuning hyperparameters over time in a supervised way.

Keywords Software effort estimation · Cross-company learning · Transfer learning · Concept drift · Online learning · Hyperparameter tuning

Communicated by: David Bowes, Emad Shihab and Burak Turhan

A preliminary version of this work appeared in Minku and Hou (2017). This work was supported by EPSRC Grant No. EP/R006660/1.

✉ Leandro L. Minku
L.L.Minku@cs.bham.ac.uk

[1] School of Computer Science, The University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK

List of Acronyms

Machine Learning Acronyms:

CC	Cross-Company
BagRT	Bagging ensembles of Regression Trees
Dycom	Dynamic Cross-company Mapped Model Learning
LogLR	Linear Regression in the Logarithmic scale
SEE	Software Effort Estimation
RT	Regression Trees
WC	Within-Company

Performance Metric Acronyms:

AE	Absolute Error
IQR	Inter-Quartile Range
LogAE	Absolute Error in the Log scale
MAE	Mean Absolute Error
MdAD	Median Absolute Deviation
MdAE	Median Absolute Error
MdLogAE	Median Absolute Error in the Log scale
SA	Standardised Accuracy
SA_{AE}	Standardised Accuracy with respect to Absolute Error
SA_{LogAE}	Standardised Accuracy with respect to Absolute Error in the Log Scale

1 Introduction

Software Effort Estimation (SEE) is the process of estimating the effort required to develop a software project. The use of machine learning for creating SEE models based on data describing completed projects has been studied for many years (Boehm 1981; Jørgensen and Shepperd 2007; Dejaeger et al. 2012; Sarro et al. 2016; Song et al. 2018). Such SEE models could form useful tools to help experts to perform and/or re-think their effort estimations. These estimations can then be used to inform many important project decisions, such as project bidding, requirements selection, task allocation, etc. Therefore, several companies in the software industry (for instance in the US and China) adopt model-based SEE methods (Menzies et al. 2017). Such methods are also widely advocated by professional societies and certification programs (see for example http://www.iceaaonline.com/, tiny.cc/gox9xy and tiny.cc/hmx9xy).

However, the process of creating SEE models faces several challenges, such as the difficulty in tuning hyperparameters[1] of machine learning approaches, the fact that SEE models may become unsuitable over time, the heterogeneity of software projects, and the high cost associated to collecting Within-Company (WC) projects for training SEE models. The challenges faced by SEE can cause several of the SEE approaches to perform poorly under certain conditions that one could encounter in practice. Research in the area has thus continued to better understand the problem and overcome these challenges, so that SEE can become more widely applicable (Sarro et al. 2016; Lokan and Mendes 2017; Sarro and Petrozziello 2018; Song et al. 2018). In particular, several studies have investigated the use

[1] In machine learning, hyperparameters are parameters that must be set before learning begins, as opposed to model parameters learnt by the algorithm.

of Cross-Company (CC) projects to reduce the cost of collecting WC projects for training SEE models (Kitchenham et al. 2007; Menzies et al. 2013; Minku and Yao 2017).

Despite the availability of potentially useful CC projects in software engineering repositories (ISBSG 2011; Menzies et al. 2017), the different processes and environments underlying different companies render CC projects potentially heterogeneous with respect to the projects being estimated (Minku 2016). Therefore, several studies proposed ways to tackle heterogeneity in CC SEE (Kocaguneli et al. 2010, 2015; Menzies et al. 2013; Minku and Yao 2014, 2017). In particular, the approach Dycom (Minku and Yao 2014) managed to drastically reduce the number of WC projects used for training while maintaining or slightly improving predictive performance in comparison with WC models. Dycom is the only approach able achieve that while treating the online learning nature of the SEE problem, i.e., the fact that new projects arrive over time and that SEE models must be able to adapt to changes that could otherwise affect their suitability (Minku and Yao 2017).

An interesting observation here is that, even though Dycom has been investigated only in the context of estimating the development effort of software projects as a whole, its general framework could also potentially be very useful for Agile estimation (Usman et al. 2014). In particular, its treatment of the estimation process as an online learning problem means that not only feedback from estimations given to previous projects can be used for improving and adapting estimates for future projects, but also feedback from estimations of previous sprints of a project could be used for improving and adapting estimations for future sprints of this project. Therefore, Dycom's framework is compatible with Agile estimation, and could potentially be used to support the improvement of Agile estimations during the course of a project. This is a direction worth exploring in the future.

In order to use CC projects for SEE, Dycom splits the available CC projects into different CC subsets containing more homogeneous projects. Each CC subset is used to build a different CC SEE model. Homogeneity is defined based on a given criterion, e.g., the productivity of the projects. Mapping functions are then learned to dynamically map the estimations given by the CC models to estimations reflecting the current *context* (in particular, the relationship between project input attributes and effort) of a given company. These mapping functions are learned by using a small number of WC training projects received over time.

However, the CC splits can have a significant impact on Dycom's predictive performance (Minku and Hou 2017), and it is unclear how to best split the CC projects into different subsets. Clustering methods could potentially help to choose good splits. However, they typically rely on pre-defining the number of clusters (CC subsets) to be created. This number can itself affect Dycom's predictive performance. Some clustering methods use unsupervised procedures to automatically tune the number of clusters. However, these procedures do not make use of predictive performance information. Therefore, it is unknown whether they could lead to good predictive performance for Dycom. Meanwhile, the literature on online learning lacks procedures for automatically choosing hyperparameters in a supervised manner, i.e., based on predictive performance information. Even if an online supervised tuning procedure was available, as Dycom is intended to reduce the number of WC projects required for training, it is unknown whether the predictive performance calculated based on such WC projects would be reliable enough for the tuning procedure to succeed.

With that in mind, the first aim of this paper is to propose a novel procedure for automatically tuning hyperparameters over time in online supervised learning. The proposed procedure makes use of predictive performance information acquired over time to dynamically select and update the most suitable number of CC subsets over time. The second aim of this paper is to investigate whether clustering methods are successful in avoiding poor CC splits for Dycom when used in combination with (1) the proposed online supervised hyperparameter tuning procedure, and (2) existing offline unsupervised procedures for tuning the number of CC subsets.

Here, *success in avoiding poor CC splits* is defined as the ability to create CC splits that enable Dycom to obtain (1) better predictive performance than a (trivial) median baseline model and than WC models, both trained on the same limited number of WC projects as Dycom, and (2) at least similar predictive performance to WC models trained on a much larger (e.g., 10 times larger) number of WC projects. How successful the approach is in terms of (1) depends on the statistical significance, effect size and magnitude of the improvements in predictive performance. How successful the approach is in terms of (2) depends on whether the approach manages to at least maintain predictive performance while using 10 times less WC training projects, based on statistical tests and effect size. More details on the approaches compared against Dycom, the performance metrics, the statistical tests and the measure of effect size used in this study are provided in Section 8.

This paper is further organised as follows. Section 2 presents the research questions answered to achieve the aims of this work, and summarises the contributions of this work. Section 3 explains related work. Section 4 explains Dycom. Section 5 explains how to use Dycom with clustering methods. Section 6 explains the procedures for automatically tuning the number of CC subsets, including the proposed online supervised hyperparameter tuning procedure. Section 7 explains the datasets used in the case study. Section 8 explains the setup of the experiments performed to achieve the aims of this work. Sections 9, 10, 11 and 12 present the analyses of the experiments. Section 13 presents a discussion regarding running time and overfitting. Section 14 discusses threats to validity. Section 15 presents the conclusions and future work.

2 Research Questions and Contributions

To achieve the aims outlined in Section 1, this paper proposes an online supervised hyperparameter tuning procedure and answers the following research questions:

- **RQ1:** Which clustering and tuning approach is the most successful in improving Dycom's predictive performance in comparison to other approaches that also have access to a limited number of WC training projects?
- **RQ2:** Why is this approach the most successful in achieving that?
- **RQ3:** Is this approach to Dycom always successful in reducing the number of WC projects required for training while maintaining predictive performance in comparison to WC models trained on more WC projects?
- **RQ4:** If not, why does it sometimes fail in achieving that? Can that give us any insights into how to use it?

A case study using the ISBSG Repository (ISBSG 2011) is performed to answer the research questions. Two of the approaches investigated with Dycom consist of Expectation-Maximisation (Bishop 2006) and X-Means (Pelleg and Moore 2000), using their built-in offline unsupervised tuning procedures. The other four consist of four other clustering methods that do not have built-in tuning procedures, used in combination with the proposed online supervised tuning procedure. These four clustering methods are: Threshold (Minku and Yao 2014), K-Means (Rokach and Maimon 2005), Hierarchical Clustering (Rokach and Maimon 2005) and Spectral Clustering (Kannan et al. 2000; Shi and Malik

2000). All approaches are investigated with three different base learners: Regression Trees (RTs), Bagging ensembles of RTs (BagRT), and Linear Regression in the Logarithmic Scale (LogLR).

The study shows that the proposed online tuning procedure in combination with a simple threshold-based clustering method was the most successful approach in avoiding poor CC splits for Dycom. In particular, it was the only Dycom approach that always outperformed both the median baseline and a corresponding WC approach trained on the same number of WC projects (RQ1). Threshold clustering performed well in comparison to more complex clustering methods because the CC training projects are exponentially distributed and not really clustered (RQ2). In most cases, it enabled Dycom to use ten times less WC training projects than a corresponding WC model, while maintaining its predictive performance. Therefore, this approach was generally successful in avoiding poor CC splits. However, in 3 out of 18 cases, it failed to reduce the number of WC training projects by 10 times while maintaining predictive performance (RQ3). The analysis found that not only the characteristics of the CC projects, but also the base learners being used can affect the number of WC projects required for training. When the CC projects were fewer and older, RT and BagRT led to SEE models whose performance varied more. This caused the proposed online supervised tuning procedure to struggle to keep track of the best number of CC subsets, potentially requiring WC training projects to be collected more frequently. LogLR led to more stable predictive performance in this scenario, facilitating the choice of the number of CC subsets by the proposed online supervised tuning procedure and avoiding the need for collecting a larger number of WC training projects (RQ4). Overall, the proposed online supervised tuning procedure was generally successful in enabling a very simple threshold-based clustering approach to obtain the most competitive results. This demonstrates the value of automatically tuning hyperparameters over time in a supervised way.

This paper provides the following novel contributions:

- The first hyperparameter tuning procedure for online supervised learning (Section 6.1).
- An investigation of how successful the SEE approach Dycom is in reducing the number of required WC training projects when used in combination with hyperparameter tuning to avoid the need for manually pre-defining the number of CC subsets (Sections 9 and 11).
- An investigation of which clustering method is the most adequate for use with Dycom when adopting hyperparameter tuning for the number of CC subsets (Section 9), and an investigation of why/when this method is expected to be adequate (Section 10).
- An investigation of Dycom using RT, BagRT and LogLR as base learners, demonstrating how well Dycom performs with these base learners and when they are recommended for use with Dycom (Sections 9 to 12).
- A detailed investigation of when and how to use the proposed online supervised hyperparameter tuning procedure in the context of SEE (Section 12).

Through the use of Dycom with online supervised hyperparameter tuning, this work is a step in the direction of better addressing four challenges: (1) high cost of collecting WC training projects; (2) heterogeneity of projects, which can hinder SEE modelling; (3) difficulty in tuning hyperparameters of machine learning approaches; and (4) changes in the data generating process (a.k.a., concept drifts), which can affect the suitability of SEE models over time. Better addressing these challenges contributes towards better applicability of machine learning-based SEE, because it becomes easier and less costly to run SEE approaches when it is not necessary to collect large bodies of WC training projects, when key hyperparameters can be automatically tuned, and when the approach can automatically adapt to changes that would otherwise cause the software effort estimations to become unsuitable over time.

A preliminary version of this work appeared in Minku and Hou (2017). However, that paper focused mainly on clustering methods without automated tuning procedures. The only clustering method investigated with automated tuning was Expectation-Maximisation. Different from that paper, the current paper has a strong focus on the use of automated tuning of the number of CC subsets, with *all* clustering methods being investigated using automated tuning. Therefore, this paper presents a completely new case study, where the only Dycom approach that has already been investigated in previous work (Minku and Hou 2017) was clustering Dycom with Expectation-Maximisation. This paper also investigates two additional clustering methods (X-Means and Spectral Clustering) which were not investigated in Minku and Hou (2017), and two additional base learners (LogLR and BagRT). This is the first work to investigate Dycom with other base learners than RTs, leading to additional useful knowledge on the benefits of Dycom for SEE.

3 Related Work

Section 3.1 presents related work on CC learning and heterogeneity in SEE. Section 3.2 presents related work on supervised hyperparameter tuning procedures in software analytics. Unsupervised hyperparameter tuning procedures are explained in Section 6, being part of the methodology to answer the research questions posed in Section 2.

3.1 CC Learning and Heterogeneity in SEE

Many studies have investigated the predictive performance of CC versus WC SEE models. A systematic literature review published by Kitchenham et al. (2007) found ten studies, seven of which were independent. Three of these concluded that the predictive performance of CC and WC models was similar, whereas the other four concluded that the predictive performance of CC models was worse. McDonell and Shepperd (2007) also performed a systematic literature review to investigate whether CC models have similar predictive performance to WC models, and found no strong evidence in support of either type of model. Ordinary Least Squares Regression, Stepwise Regression, Robust Regression, RTs and k-Nearest Neighbours were among the machine learning approaches investigated.

Several of these studies used CC projects in an attempt to completely eliminate the need for WC projects (Kitchenham et al. 2007; McDonell and Shepperd 2007). They compared WC models against models created only with projects from other companies (e.g., Briand et al. 2000; Wieczorek and Ruhe 2002). Some other studies used both CC and WC projects in at least part of the procedure for building SEE models (Jeffery et al. 2010; Kitchenham and Mendes 2004; Lefley and Shepperd 2003). For example, Lefley and Shepperd (2003) trained SEE models with both CC and WC projects, in an attempt to overcome the small number of WC projects (Lefley and Shepperd 2003).

Both systematic reviews mention that the level of heterogeneity of the single-company being estimated may vary and influence the WC models' performance. This has influenced more recent studies that use local learning approaches to tackle the heterogeneity of WC and CC projects. For example, Kocaguneli et al. (2010) used a tree-based filtering mechanism, obtaining very encouraging results. Models trained solely with different types of projects (or projects from different centres of a company) from the ones being estimated obtained overall similar performance to models trained on projects of the same type (or from the same centre). Turhan and Mendes (2014) investigated the use of a filtering mechanism based on k-Nearest Neighbours, obtaining very encouraging results in the area of web effort estimation. Specifically, filtered Stepwise Regression CC models achieved similar predictive performance to WC models in seven out of eight datasets, and worse predictive performance in only one dataset.

Menzies et al. (2013) proposed to cluster WC+CC projects to tackle heterogeneity using a method called WHERE. This method splits projects according to the input attributes of highest variability. Prediction rules are then created for each cluster based on a method called WHICH. Given a certain cluster, its neighbouring cluster with the lowest required efforts was referred to as the *envied* cluster. When making predictions for WC projects belonging to a cluster, rules created using only the CC projects from the envied cluster were better than rules created using only the WC projects from the envied cluster. So, the authors recommended to cluster WC+CC projects, but to learn rules using solely the CC projects from the envied cluster. This study was in the context of both SEE and software defect prediction, but this particular conclusion was based only on the defect prediction data.

Other studies also investigated the use of clustering methods to tackle heterogeneity. For example, Huang et al. (2008) investigated K-Means and Scheffe's method to cluster CC projects based on their input attributes. An Ordinary Least Squares model was created for each cluster. When estimating a new project, the cluster to which this project belongs is determined and its corresponding SEE model is used for performing the estimation. The predictive performance obtained using clustering was similar to that obtained without clustering. Gallego et al. (2007) partitioned CC projects into different subsets based on certain input attributes of interest. Each of the partitions was then further clustered using Expectation-Maximisation. Linear or Exponential Regression models were created for each cluster. Predictions were made in a similar way to Huang et al. (2008)'s method. The authors concluded that clustering can improve predictive performance, even though no statistical tests were used in this comparison. These methods were evaluated for making predictions for CC projects, i.e., no WC dataset was used in these studies.

It is important to note that, even though CC/WC projects could be considered as potentially more/less heterogeneous with respect to the projects being estimated, WC projects may also be heterogeneous. Therefore, the terms CC/WC should not be considered as synonyms of heterogeneous/homogeneous (Minku 2016), and the possible heterogeneity of WC projects should be tackled. Kocaguneli et al. (2012) and Minku and Yao (2013) investigated the use of tree-based SEE models to tackle heterogeneity in general, i.e., not restricted to CC projects. Other local approaches such as k-Nearest Neighbours (Shepperd and Schofield 1997; Amasaki et al. 2011) could also be seen as tackling heterogeneity.

The studies above did not take into account the fact that SEE is an online learning problem, where new projects need to be predicted over time, and changes suffered by the company may affect the quality of existing SEE models (Minku and Yao 2012, 2017). With that in mind, Dynamic Cross-company Learning (Minku and Yao 2012, 2017) creates an ensemble of WC and CC models and dynamically identifies which of them is currently beneficial for SEE. The beneficial models are emphasised when the ensemble is asked to estimate a new WC project. This approach managed to improve predictive performance in comparison with a WC model. However, Dynamic Cross-company Learning can only benefit from CC projects when they match the current WC context reasonably well. It still requires a fair amount of WC projects to achieve good performance during the periods when the CC models are not beneficial.

Kocaguneli et al. (2015) investigated a tree-based filtering mechanism called TEAK (Kocaguneli et al. 2012) to tackle heterogeneity. This mechanism creates trees to represent training projects and provide effort estimations. CC or WC training projects corresponding to sub-trees of high variance are assumed to be detrimental and filtered out. Besides investigating TEAK for CC web effort estimation, Kocaguneli et al. (2015) also investigated its ability to transfer knowledge from the past to the present in conventional WC SEE. The approach managed to obtain similar performance when using only training projects from the same time period as the project being estimated, and when using a mix of training projects from the same and different time periods. However, similar to Dynamic Cross-company Learning (Minku and Yao 2012, 2017), this approach still relies on a good number of projects from the same time period to be available for the process of generating the SEE model.

The approach Dycom (Minku and Yao 2014) was proposed to overcome this limitation. Similar to Dynamic Cross-company Learning, it maintains an ensemble of WC and CC models. However, instead of just identifying which past WC or CC models are more beneficial, it maps the estimations given by the CC models to the WC context. In this way, it can significantly reduce the number of WC projects needed for training while maintaining or slightly improving predictive performance in comparison with a WC model. However, Dycom requires CC projects to be split into different subsets to create its CC models, and the choice of CC split can significantly affect its predictive performance. As explained in Section 1, this paper aims to investigate which clustering methods and automated hyperparameter tuning procedures can help with that. Section 4 explains Dycom in more detail.

3.2 Supervised Hyperparameter Tuning in Software Analytics

Supervised machine learning approaches are frequently sensitive to the choice of hyperparameter values (Lavesson and Davidsson 2006; Song et al. 2013; Nair et al. 2018). For example, in the context of online SEE, a study (Song et al. 2013) showed that Multilayer Perceptrons (Bishop 2006) are highly affected by hyperparameter choice, specially when used as single learners. The use of Bagging (Breiman 1996) to combine multiple Multilayer Perceptrons into ensembles helped to decrease their sensitivity to hyperparameters, but hyperparameter choice still had a significant effect. The preliminary study leading to the current paper (Minku and Hou 2017) also showed that Dycom (Minku and Yao 2014) is sensitive to the number of CC subsets. Conversely, WEKA's REPTree RTs (Hall et al. 2009), and especially Bagging ensembles of REPTrees, tended to perform well with default hyperparameter values (Song et al. 2013).

Even though the studies above show that online supervised learning for SEE is affected by hyperparameter choice, there is no agreed procedure to tune hyperparameters in the online supervised learning literature. The difficulty in tuning hyperparameters for online supervised learning algorithms is due to the fact that there may not be data beforehand to evaluate different hyperparameter choices and identify which of them is most adequate. Moreover, even if there is an initial portion of data, the best hyperparameter values for this initial portion may not be the same as the best ones for later periods of time, because the data generating process may change over time (Minku and Yao 2014; Ditzler et al. 2015). No general purpose hyperparameter tuning procedure for online supervised learning is available to dynamically and automatically tune hyperparameters at runtime.

Different from the literature on online supervised learning, the literature on offline supervised learning has several studies investigating procedures for tuning hyperparameters of machine learning approaches (Nair et al. 2018). For example, in the context of SEE, Oliveira et al. (2010) investigated the use of a Genetic Algorithm to automatically tune hyperparameters and select input attributes for several offline supervised learning algorithms. The search for good hyperparameter values is guided by their resulting predictive performance on the training set. It is then hoped that these values will also lead to good predictive performance on the test set. The study investigated this tuning procedure with the following offline supervised learning approaches: Support Vector Regression, Multilayer Perceptrons and M5 model trees. This procedure was evaluated on six datasets and shown to lead to both improvements in predictive performance and reduction in the complexity of the resulting predictive model.

Corazza et al. (2013) investigated the use of Tabu Search to tune the hyperparameters of (offline) Support Vector Regression in the context of SEE. The search for good hyperparameter values is guided by their resulting validation predictive performance, calculated through the cross-validation procedure. Cross-validation is run based on all data that are not used for testing. The advantage of using validation predictive performance over training predictive performance is that it works as an estimate of the predictive performance on data not used for training. This can avoid overfitting and potentially result in better predictive performance on the test set. An investigation based on 21 datasets found that Tabu Search led to significantly better predictive performance than random hyperparameter values, default values provided by WEKA (Hall et al. 2009), and grid-search, with very few exceptions. In particular, the Median Absolute Error obtained using the Support Vector Regression model with the best estimates without Tabu Search was on median about one half the error obtained by the model tuned with Tabu Search.

More recently, Xia et al. (2018) investigated the use of Differential Evolution to tune the hyperparameters of (offline) Analogy-Based SEE. Their instance of the Differential Evolution algorithms was given a small budget of model evaluations to find the hyperparameter values that lead to the best Magnitude of the Relative Error or Standardised Accuracy on a data set set not used for training. They found that their method led to significantly better results than untuned Analogy-Based SEE and a baseline called Automatically Transformed Linear Model. Differences in Standardised Accuracy with respect to untuned Analogy-Based SEE varied from 17% to 27%. Xia et al. (2018) further investigated Non-dominated Sorting Genetic Algorithm II and Multiobjective Evolutionary Algorithm Based on Decomposition to tune hyperparameters. They showed that time-consuming optimisers such as Non-dominated Sorting Genetic Algorithm II did not lead to improvements in Standardised Accuracy in comparison to faster optimisers such as Differential Evolution using a small budget of model evaluations.

Tantithamthavorn et al. (2016) investigated the impact of hyperparameter tuning in software defect prediction based on a case study with 18 datasets and 26 offline supervised learning algorithms. They tuned hyperparameters based on 100 repetitions of the out-of-sample bootstrap procedure. The tuned hyperparameters led to improvements of up to 40 percentage points in the Area Under the Curve. This study was later on extended, including three more hyperparameter tuning procedures (Random Search, Genetic Algorithms and Differential Evolution) and several performance metrics (Tantithamthavorn et al. 2018). They found that different hyperparameter tuning procedures led to similar benefits in terms of performance improvement when applied to software defect prediction. They emphasised that, even though hyperparameter tuning can lead to large performance improvements, this depends on the machine learning algorithm being tuned. Some machine learning algorithms such as Random Forests were not so much affected by hyperparameter choice in software defect prediction. This complements Song et al. (2013)'s study, which found that Bagging ensembles of REPTrees were not much affected by hyperparameter choice in the context of SEE.

Agrawal and Menzies (2018) investigated the use of Differential Evolution to automatically tune hyperparameters used by the SMOTE offline learning algorithm in the context of software defect prediction. Their Differential Evolution attempts to find the hyperparameter values that lead to the best predictive performance on a validation set. This validation set is a portion of the data that is not used for training or testing the predictive model. Their experiments, which were based on seven datasets, showed that Differential Evolution can lead to large improvements in predictive performance (area under the curve increased by up to 60%) when tuning SMOTE's hyperparameters. Differential evolution is also applicable to other offline machine learning algorithms (Fu et al. 2016).

Fu and Menzies (2017) applied Differential Evolution as a hyperparameter tuner to (offline) Support Vector Machines for the problem of finding which questions in the Stack Overflow programmer discussion forum can be linked together. Their Differential Evolution attempts to find the hyperparameter values that lead to the best F1-score on a validation set (called tuning data in their work). Their approach usually led to similar or better results than a state-of-the-art Deep Learning Convolutional Neural Network in terms of precision, recall and F1-score, while being 84 times faster to run.

Overall, even though several studies emphasize the importance of tuning hyperparameters of machine learning approaches in software analytics, and several hyperparameter tuning procedures have been used in offline learning scenarios in this context, there is no hyperparameter tuning procedure for online supervised learning.

4 Dycom

Dycom (a.k.a. Dynamic Cross-company Mapped Model Learning) is an SEE online ensemble learning approach which aims at reducing the number of WC projects required for training, while ensuring that the ensemble is updated to reflect potential changes in the underlying data generating process of a given company. *This approach achieves that by using a small number of WC training projects received over time to learn how to map the effort estimations given by CC models to the current context of the single company in which we are interested.* It is based on the observation that there is a relationship between the SEE context of a given company and other companies. Minku and Yao (2014) formalise the relationship between two companies C_A and C_B as follows:

$$f_A(\mathbf{x}) = g_{BA}(f_B(\mathbf{x})) \tag{1}$$

where C_A is the company in which we are interested; C_B is another company (or a department of this other company); f_A and f_B are the true functions providing C_A's and C_B's required efforts, respectively; g_{BA} is a function that maps the effort from C_B's context to C_A's context; and $\mathbf{x} = [x_1, x_2, \cdots, x_n]$ are the input attributes describing a software project. The functions f_A, f_B and g_{BA} can be of any type. An illustrative example where $f_A(\mathbf{x}) = g_{BA}(f_B(\mathbf{x})) = 1.2 \cdot f_B(\mathbf{x})$ can be found in Minku and Yao (2014).

Given (1), the task of learning an SEE model for a company C_A can involve the task of learning the relationship between C_A and other companies. Therefore, Dycom uses CC training projects to learn one or more CC models, and uses a very limited number of WC training projects to learn a function that maps the estimations given by each CC model to estimations in the WC context. It is hoped that the task of learning the mapping functions is less difficult than the task of learning a whole WC model based solely on the WC training projects, as this would considerably reduce the amount of WC projects required for learning. As summarised by Minku et al. (2015), Dycom works as follows.

CC training projects: Dycom uses CC training projects that are available beforehand. Past CC projects were found to be beneficial for prolonged periods of time (e.g., 10 years), even though these periods of time are typically intercalated with periods where such projects are detrimental (Minku and Yao 2017). Therefore, so long as the beneficial and detrimental CC projects are properly distinguished and mapped to the context of the company being estimated, CC training projects available beforehand are of practical use. In particular, acquiring additional CC projects over time was found to be beneficial only in cases where the initial CC dataset was small (Minku 2018). If one wishes to use Dycom with additional CC projects, it can be reset and retrained with those projects. Such reset does not need to be performed often, given the prolonged periods of time where CC projects can be beneficial.

CC training project subsets: the original Dycom study split CC projects into M different training subsets C_{Bi}, $1 \leq i \leq M$ based on productivity thresholds (Minku and Yao 2014). For example, if the productivity of the CC training projects in terms of effort (in person-months) divided by size varies from 3 to 38, one may wish to separate these projects into three subsets, one containing projects with productivity below 10, one containing projects with productivity between 10 and 15, and one for projects with productivity higher than 15. Each subset C_{Bi} is considered as a separate CC training set. Dycom can also use clustering methods to create the CC splits, as further explained in Section 5. The reason for splitting CC projects will be explained later in the paragraph on mapping functions. Note that we use the term CC loosely herein. For example, projects from different departments within the same company could be considered as CC projects if such departments employ largely different practices.

CC SEE models: Each of the M CC training sets is used to create a different CC model $\hat{f}_{Bi}$, $1 \leq i \leq M$, using a given base learning algorithm.

WC training projects: Dycom considers that the WC projects arrive in order of completion, i.e., online. Each moment in time when a new WC project arrives is referred to as a *time step*. Even though a new WC project needs to be estimated at each time step, information on the true effort required to develop the project is assumed to be collected for only a small portion of the WC projects. This is because such information is expensive to collect (Kocaguneli et al. 2013). Only the WC projects for which the true required effort is collected can be used as WC training projects.

Mapping functions: Whenever a new WC training project arrives, each model $\hat{f}_{Bi}$ is asked to perform an SEE. Each SEE is then used to create a mapping training example $(\hat{f}_{Bi}(\mathbf{x}), y)$. A mapping function $\hat{g}_{BiA}$ that receives estimations $\hat{f}_{Bi}(\mathbf{x})$ in the context of C_{Bi} as input and maps them to estimations in the context of C_A is trained with the mapping training example. Dycom considers that the relationship formalised in (1) can be modelled reasonably well by linear functions of the format $\hat{g}_{BiA}(\hat{f}_{Bi}(\mathbf{x})) = \hat{f}_{Bi}(\mathbf{x}) \cdot b_i$

when different subsets containing relatively more similar CC training projects are considered separately. This is the reason to split CC training projects into different subsets.

Learning a function of the format $\hat{g}_{BiA}(\hat{f}_{Bi}(\mathbf{x})) = \hat{f}_{Bi}(\mathbf{x}) \cdot b_i$ is equivalent to learning the factor b_i. This is done using (2):

$$b_i = \begin{cases} 1, \quad \text{if no mapping training example has been received yet} \\ \dfrac{y}{\hat{f}_{Bi}(\mathbf{x})}, \text{ if } (\hat{f}_{Bi}(\mathbf{x}), y) \text{ is the first mapping training example} \\ lr \cdot \dfrac{y}{\hat{f}_{Bi}(\mathbf{x})} + (1 - lr) \cdot b_i, \text{ otherwise.} \end{cases} \tag{2}$$

where $(\hat{f}_{Bi}(\mathbf{x}), y)$ is the mapping training example being learnt, lr $(0 < lr < 1)$ is a pre-defined smoothing factor and the factor b_i in the right side of the equation is the previous value of b_i. This equation enables the mapping function to dynamically adapt to potential changes affecting software effort over time. The fact that a dynamically adaptive mapping function is used means that Dycom can dynamically adapt to changes in the context of the company being estimated even considering that its CC models are not updated over time. For a more detailed explanation of this equation, we refer the reader to Minku and Yao (2014).

The factor b_i represents the relationship between the effort required by a given company and that required by other companies. Therefore, it can be used to monitor how this relationship changes over time (Minku and Yao 2014). In this way, it is possible to check the effectiveness of strategies to improve the productivity of a company.

WC SEE model: Whenever a new WC training project arrives, it is used to train a WC model $\hat{f}_{W_A}$. This model is not expected to perform very well, because it will be trained on a limited number of projects. However, its effort estimations may be helpful when used in an ensemble together with the mapped estimations, given that ensembles have been showing to improve SEE considerably (Minku and Yao 2013, 2012, 2017). It is also worth noting that, despite being potentially more homogeneous than CC projects, the WC projects could possibly also present a significant level of heterogeneity. In order to tackle this heterogeneity, it is recommended to use Dycom with base learners that are local approaches, such as RTs (Minku and Yao 2013).

Dycom's SEEs: Both the mapped models $\hat{g}_{BiA}(\hat{f}_{Bi})$ and the WC model $\hat{f}_{W_A}$ can provide an SEE in the WC context when required. The SEE given by Dycom is the weighted average of these $M + 1$ estimations:

$$\hat{f}_A(\mathbf{x}) = \left[\sum_{i=1}^{M} w_{Bi} \cdot \hat{g}_{BiA}(\hat{f}_{Bi}(\mathbf{x}))\right] + w_{W_A} \hat{f}_{W_A}(\mathbf{x}), \tag{3}$$

where the weights w_{Bi} and w_{W_A} represent how much we trust each of the models, are positive and sum to one. So, Dycom uses an ensemble of mapped and WC SEE models.

Weights: The weights are initialised so that they have the same value for all models being used in the ensemble and are updated as as follows: whenever a new WC training project is made available, the model which provided the lowest absolute error is considered to be the *winner* and the others are the *losers*. The losers have their weights multiplied by a pre-defined hyperparameter β $(0 < \beta \leq 1)$, and then all weights are normalised in order to sum up to one. In this way, the weights can distinguish between CC models that are currently beneficial and detrimental to the single company being estimated. In particular, β gives exponentially less importance to models that perform less well on the WC training projects. Such exponential decay enables poor models to be quickly identified in the presence of a small number of WC training projects. It is therefore a key feature of Dycom to save the cost of collecting WC training projects.

Algorithm 1 presents Dycom's learning process, and Fig. 1 illustrates it with a diagram. Dycom first splits the CC projects based on a CC splitting algorithm (e.g., split based on productivity thresholds) and learns the CC models based on a supervised base learning algorithm (lines 1 to 4). The weight associated to each CC model is initialised to $1/M$, so that each model has equal weight and all weights sum to one (line 5). The mapping functions are initialised to $b_i = 1$ (line 6). Before any WC training project is made available, the weight w_{W_A}corresponding to the WC model is initialised to zero (line 8), because this model has not received any training yet. In this way, CC models can be used to make predictions while there is no WC training project available (line 9). After that,

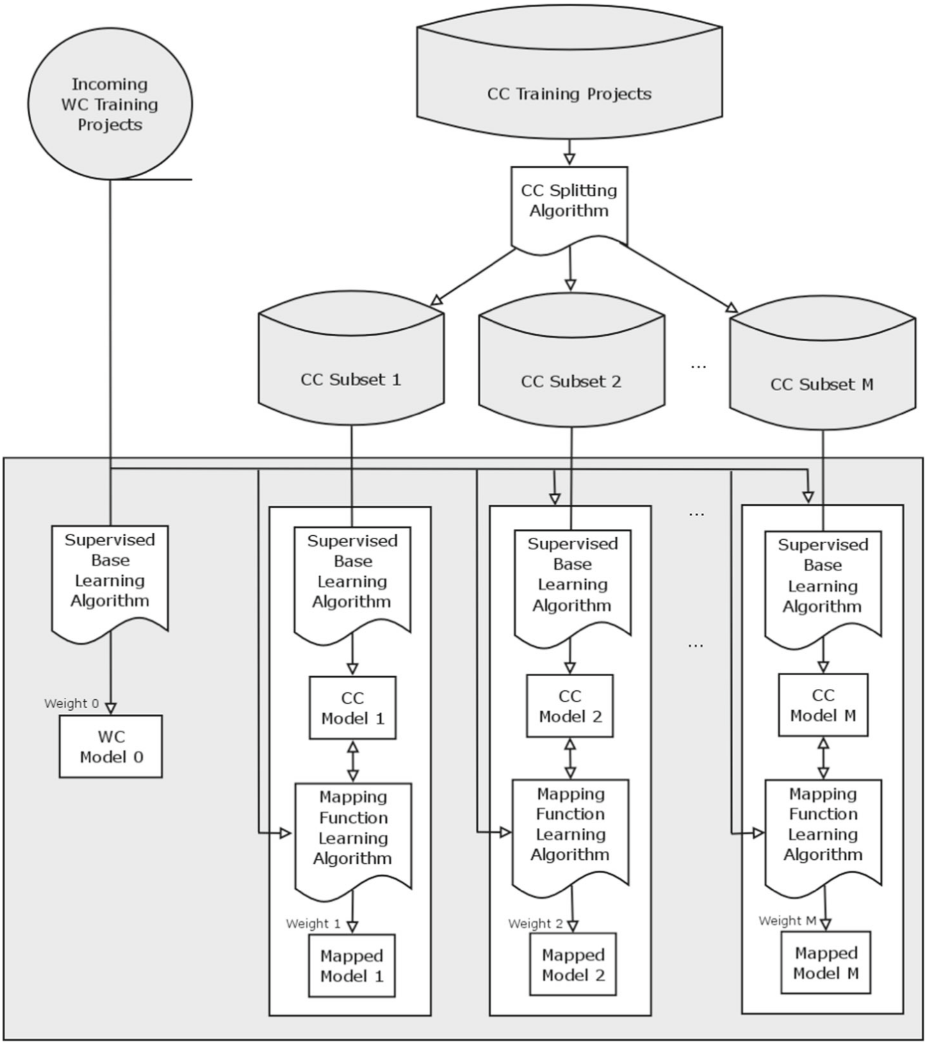


Fig. 1 Dycom's learning process

for each new WC training project, the weights are updated (lines 12–21). In particular, if the WC training project is the first one, the weight of the WC model needs to be set for the first time to a value different from zero (line 19). After updating the weights, the mapping training examples are created and used to learn (update) the corresponding mapping functions based on an algorithm that implements (2) (lines 24 and 25). Finally, the WC model is updated with the WC training project using a supervised base learning algorithm (line 28).

Algorithm 1 Dycom's learning process (Minku and Yao 2014).

```
Parameters:
$lr$: smoothing factor for the mapping function
$\beta$: factor for decreasing model weights
Base learning algorithm
Algorithm to split CC projects
 1: Create the CC training subsets $D_{Bi}$ ($1 \leq i \leq M$) using the algorithm to split CC projects.
 2: {Learn CC base learners:}
 3: for each CC training set $D_{Bi}$ do
 4:     Create CC model $\hat{b}_{Bi}$ using $D_{Bi}$ and the base learning algorithm
 5:     $w_{Bi} = \frac{1}{M}$ { Initialise weight.}
 6:     $b_i = 1$ { Initialise mapping function.}
 7: end for
 8: $w_{W_A} = 0$ { Initialise WC model weight.}
 9: Enable predictions to be made based on (3).
10: for each new WC training project ($\mathbf{x}$, $y$) do
11:     {Update weights:}
12:     for each model $\hat{f}_{Bi}$ and $\hat{f}_{W_A}$ do
13:         Determine the model's estimation to $\mathbf{x}$.
14:         Calculate the absolute error $AE_{Bi}$ (or $AE_{W_A}$).
15:     end for
16:     Determine loser models based on their $AE$.
17:     Multiply loser models' weights by $\beta$.
18:     if ($\mathbf{x}$, $y$) is the first WC training project then
19:         $w_{W_A} = \frac{1}{M+1}$
20:     end if
21:     Divide each weight by the sum of all weights.
22:     {Update mapping functions:}
23:     for each model $\hat{f}_{Bi}$ do
24:         Create mapping training example ($\hat{f}_{Bi}(\mathbf{x})$, $y$).
25:         Use ($\hat{f}_{Bi}(\mathbf{x})$, $y$) and $lr$ to update $\hat{g}_{BiA}$ based on (2).
26:     end for
27:     {Update WC model:}
28:     Update WC model $\hat{f}_{W_A}$ using ($\mathbf{x}$, $y$) and the base learning algorithm.
29: end for
```

5 Clustering Dycom

CC projects could potentially be split into M different CC subsets based on clustering algorithms, rather than pre-defined productivity thresholds. For that, each cluster of CC projects can be considered as a CC subset. Using clustering with Dycom offers potential advantages over pre-defined productivity thresholds. Clustering algorithms could automatically find out which CC projects are most similar to each other and group them together. In this way, project managers do not need to analyse the CC projects and their productivity to determine good productivity thresholds. A good clustering algorithm may be able to further boost Dycom's predictive performance, or avoid low predictive performance resulting from poor thresholds.

We referred to the use of clustering with Dycom as Clustering Dycom in our preliminary study (Minku and Hou 2017). However, as the original method for creating CC splits is adopted with automatically defined thresholds rather than pre-defined ones, it will also be referred to as a clustering method in this paper. Therefore, for convenience, we will always use the term Dycom instead of Clustering Dycom in this paper.

Different features can be used to describe training projects for clustering. For instance, the following three different sets of clustering features could potentially be used:

1. Productivity, measured in terms of effort divided by size. This clustering feature was used in the original Dycom study (Minku and Yao 2014), leading to good results.
2. Size and effort. Together, size and effort represent the productivity associated to a project. However, two projects with similar productivities could still have very different sizes. Therefore, using size and effort as clustering features is not the same as using productivity, and worth being investigated separately.
3. All project input and output attributes. This could potentially lead to a more detailed characterisation of the projects to be split.

It is worth noting that it is ok to use the effort as (part of) a feature for clustering, because (a) clustering is applied only to the CC *training* projects, and (b) Dycom does not require to determine the cluster to which a WC project being estimated belongs.

Our preliminary study (Minku and Hou 2017) found that the clustering features (2) and (3) above did not lead to good results as clustering features for Dycom. Therefore, this study will concentrate on productivity as a clustering feature. Six different clustering methods are investigated: Threshold Clustering, K-Means (Rokach and Maimon 2005), Hierarchical Clustering (Rokach and Maimon 2005), Spectral Clustering (Kannan et al. 2000; Shi and Malik 2000), Expectation-Maximisation (Bishop 2006) and X-Means (Pelleg and Moore 2000). These clustering methods, as well as the reasons for choosing them for the investigation, are explained below.

5.1 Threshold Clustering

We will refer to the original rule used by Dycom (Minku and Yao 2014) to create CC subsets as Threshold Clustering when the CC subsets and their corresponding productivity thresholds are determined based on Algorithm 2.

Algorithm 2 Threshold clustering.

```
Parameters:
$M$: number of clusters to be generated
$N \leftarrow$ number of CC projects to be split.
$cc\_projects \leftarrow$ list containing all CC projects to be split, sorted according to productivity.
for $c = 1$ to $M$ do
    $cc\_subsets_c \leftarrow$ first $\lfloor N/M \rfloor$ CC projects from $cc\_projects$.
    Remove first $\lfloor N/M \rfloor$ CC projects from $cc\_projects$.
    if $c < M$ then
        $thresholds_c$ = average between productivity of the last project in $cc\_subsets_c$ and first project in $cc\_projects$.
    end if
end for
Add all remaining CC projects from $cc\_projects$ into $cc\_subsets_M$.
Return $cc\_subsets$ and $thresholds$.
```

This clustering method was included in the experiments because it is based on Dycom's original rule for separating CC projects into subsets (Minku and Yao 2014).

5.2 K-Means

K-Means (Rokach and Maimon 2005) is an iterative clustering method that tries to minimise the distance of the projects to their cluster centres. It first randomly assigns projects to a pre-defined number k of clusters, where k equals to the desired number M of CC subsets to be generated. The centre of each cluster is calculated as the mean of all projects within that cluster. Then, in each iteration, projects are reassigned to the clusters with the closest centres, and the clusters' centres are re-computed. This iterative procedure is typically halted when projects stop moving between clusters, or when a maximum number of iterations is reached.

Different distance measures can be used to calculate how close a project $\mathbf{x}$ is to a cluster centre $\mathbf{c}$. In this work, the Manhattan distance was adopted:

$$D(\mathbf{x}, \mathbf{c}) = \sum_{i=1}^{F} |x_i - c_i|,$$

where F is the number of clustering features, and x_i and c_i represent the ith clustering feature of project $\mathbf{x}$ and of cluster centre $\mathbf{c}$, respectively. As Manhattan distance is affected by the scale of the features, all numeric clustering features are normalised to be within [0, 1]. As explained in the beginning of Section 5, a single numeric feature (productivity) is used in this work.

This clustering method has been chosen because it is one of the most popular and well known clustering methods in the literature.

5.3 Hierarchical Clustering

Hierarchical clustering methods (Rokach and Maimon 2005) build a hierarchy of clusters by putting together projects that are similar to each other. In this work, we have used an agglomerative hierarchical method. This method uses a bottom-up approach in which each project is considered to be a cluster in the lowest level of the hierarchy. Then, at each step of the algorithm, the two clusters that are closest to each other are merged together. In the end of the procedure, a single cluster containing all the projects is created.

In order to use the clusters provided by this method, the level of the hierarchy corresponding to the desired number M of CC subsets has to be specified. Different ways to measure the similarity between clusters can be used. In this work, the distance between two clusters is defined as the Manhattan distance between the two most distant projects, one from each cluster.

This clustering method has been chosen because hierarchical SEE models have obtained promising results for tackling heterogeneity (Kocaguneli et al. 2012; Minku and Yao 2013). Hierarchical clustering also has the advantage of being a deterministic method, i.e., it will always retrieve the same clusters when the same projects and clustering features are used.

5.4 Spectral Clustering

Spectral Clustering is a set of advanced clustering methods based on the spectrum (eigenvalues) of the similarity matrix between instances. In this work, the WEKA version implemented by Luigi Dragone http://www.luigidragone.com/software/spectral-clusterer-for-weka/ was adopted. This version is based on the work of Kannan et al. (2000) and Shi and Malik (2000).

This method considers the instances to be clustered as nodes in a graph. The weight of each edge represents the similarity between two instances. The method's goal is then to find the minimum normalised cut in the graph. This cut is, roughly speaking, a cut that partitions the graph into two subgraphs whose edges connecting them have low weights. Eigenvalue decomposition techniques are used to find the minimum normalised cut. The method then recursively cuts each of the subgraphs. The cutting process stops when it cannot find a cut that has a value below a hyperparameter α^*, $0 \leq \alpha^* \leq 1$. This means that the hyperparameter α is linked to the number of clusters that will be generated. In the end of the cutting process, each subgraph is considered to be a different cluster. As with K-Means and Hierarchical Clustering, Manhattan Distance was used as the similarity measure.

This method has been chosen for being an advanced clustering method that obtained promising results in the machine learning literature (Kannan et al. 2000; Shi and Malik 2000), and whose working mechanism is quite different from the other clustering methods investigated in this work.

5.5 Expectation-Maximisation

Expectation-Maximisation (Bishop 2006) is a density-based clustering method, which assumes that the projects belonging to each cluster are drawn from a specific probability distribution. This method thus tries to identify the clusters and their probability distributions.

Given a number of clusters M, Expectation-Maximisation first randomly assigns random values for the parameters θ_j of the probability distributions associated to each cluster C_j. It then performs two steps: expectation and maximisation. In the expectation step, the posterior probability $p(C_j|\mathbf{x})$ that a given project $\mathbf{x}$ belongs to a given cluster C_j is estimated based on the current parameters of the probability distributions. In the maximisation step, the parameters that maximise the log likelihood of the projects $ln\ p(\mathbf{x}|\theta)$ are calculated (note that $p(C_j|\mathbf{x})$ is used to compute that) and used to replace the previous parameters. The algorithm iterates through the expectation and maximisation steps until a stopping criterion is met. This could be when a pre-defined maximum number of iterations is reached, or when the the changes in the log likelihood $ln\ p(\mathbf{x}|\theta)$ become smaller than a threshold. The distributions are assumed to be Gaussian for numeric clustering features.

This algorithm has been chosen because it is associated to a potentially useful offline unsupervised procedure to automatically determine the number of clusters. This procedure is explained in Section 6.2.

5.6 X-Means

X-Means (Pelleg and Moore 2000) is a clustering method proposed to accelerate K-Means (Rokach and Maimon 2005). Each iteration of K-Means has to go through each project to determine which centre is the closest to (i.e., owns) that project. This can be a time consuming process when the number of projects is large. X-Means uses a tree-like structure to create a hierarchy of projects and reduce the number of projects that have to be compared against cluster centres. This structure is used to recursively rule out centroids as owners of any project represented by a given node or its children. Once it is determined that there is only one possible centroid to own a given project, this centroid is automatically considered to own all the children of that project as well, accelerating the algorithm.

The acceleration achieved by X-Means is not necessarily an advantage of this method for SEE, given that the number of projects is typically not large enough to make K-Means' running time a concern. However, X-Means also contains a built-in mechanism to decide the number of clusters to be used. This automated tuning procedure is presented in Section 6.3, and is the main reason to investigate X-Means in this study.

6 Procedures for Automatically Tuning Dycom'S Number of CC Subsets

As Dycom is sensitive to the number of CC subsets to be used (Minku and Hou 2017), a procedure for automatically choosing this number is desirable. However, Dycom is an online supervised learning approach, and no existing general methods for automatically tuning hyperparameters of online supervised learning approaches are available. Therefore, this section proposes a novel procedure for automatically tuning hyperparameters in an online supervised manner. This procedure is presented in Section 6.1. It benefits from predictive performance information acquired over time, so that the best number of CC subsets can be determined and updated over time.

Despite having the advantage of being a dynamic adaptive procedure with an explicit goal of choosing the number of CC subsets that leads to better predictive performance, the number of WC training projects available to compute predictive performance over time when using Dycom is small. This is because Dycom's main goal is to reduce the number of WC projects required for training. Therefore, it is unknown whether an online supervised hyperparameter tuning procedure would be suitable for Dycom. This paper will investigate that as part of research question RQ1 and RQ3, posed in Section 2.

In addition to investigating the proposed online supervised tuning procedure, this paper will also investigate the use of two offline unsupervised tuning procedures. These procedures are presented in Sections 6.2 and 6.3, and are linked to the clustering algorithms Expectation-Maximisation and X-Means, respectively. Even though Dycom is an online supervised learning approach (it is able to learn new WC training projects over time), the clustering procedure used for its CC projects is offline and unsupervised (see Section 5).

Therefore, it is possible to use existing offline unsupervised procedures to decide the number of CC subsets.

The advantage of using offline unsupervised procedures is that they can determine the number of CC subsets without requiring access to the true effort of the WC projects. The disadvantage is that they do not take Dycom's predictive performance into account, not having an explicit target of choosing the number of CC subsets that leads to better predictive performance. This paper investigates the suitability of such tuning procedures as part of RQ1.

6.1 Proposed Online Supervised Tuning Procedure

Offline supervised tuning procedures use a pre-existing dataset with known output attribute values to estimate the predictive performance obtained by different hyperparameter choices. In the case of SEE, the known output attribute values are the true software required efforts. Such strategy is not applicable to online supervised learning algorithms. This is because (1) a dataset with known output attribute values may not be available before training starts, and (2) even it it was, the best hyperparameter values for this dataset may not be the best for later periods of time, due to changes that the data generating process may suffer. This section thus proposes a novel hyperparameter tuning procedure that does not require a pre-existing dataset with known output attribute values, being applicable to online supervised learning approaches such as Dycom.

The general idea of the proposed procedure is to maintain a number of Dycom instances created using different numbers of CC subsets, and select to provide effort estimations only the one which currently has the smallest validation error. The scheme of the procedure when applied to Dycom is shown in Fig. 2. The pseudocode is shown in Algorithm 3.

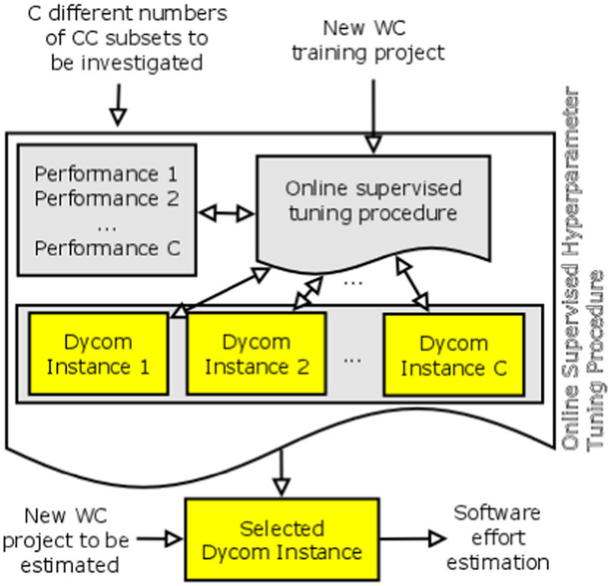

C different numbers of CC subsets to be investigated
New WC training project
Performance 1
Performance 2
...
Performance C
Online supervised tuning procedure
...
Dycom Instance 1
Dycom Instance 2
...
Dycom Instance C
Online Supervised Hyperparameter Tuning Procedure
New WC project to be estimated
Selected Dycom Instance
Software effort estimation

Fig. 2 Online supervised hyperparameter tuning procedure applied to Dycom. The procedure chooses an instance of Dycom, trained based on a given number of CC subsets, to be used for predictions

Algorithm 3 Online supervised hyperparameter tuning procedure applied to Dycom.

```
Parameters:
$C$: number of CC subsets to be investigated
All parameters required for running Dycom (see Algorithm 1)
1: Initialise Dycom instances $Dycom_1$ to $Dycom_C$.
2: for each new WC training project $(\mathbf{x}, y)$ do
3:     Get the predictions given by $Dycom_1(\mathbf{x})$ to $Dycom_C(\mathbf{x})$.
4:     Use these predictions to update the estimates of the current predictive performance
       of each Dycom instance.
5:     Train $Dycom_1$ to $Dycom_C$ using $(\mathbf{x}, y)$.
6:     Select the Dycom instance with the best estimated current predictive performance
   for predicting new WC projects.
7: end for
```

The proposed procedure enables learning and hyperparameter tuning to occur concurrently by maintaining C different instances of Dycom, created using C different pre-defined numbers of CC subsets. As the number of CC projects available for training is typically not very large, it is possible for the tuning procedure to cover a wide range of values, enabling good hyperparameter choices to be found. In particular, one could consider that a CC SEE model trained with less than 10 projects is unlikely to perform well. Therefore, if N CC training projects are available, one could specify all CC numbers from 1 to $\lfloor N/10 \rfloor$ to be considered by the tuning procedure, i.e., $C = \lfloor N/10 \rfloor$. Even though the different CC subsets may not have an equal number of CC projects, this strategy could work as a good rule of thumb to enable a wide range of numbers of CC subsets to be investigated.

For example, the ISBSG dataset with the maximum number of CC projects used in this study had 224 CC projects. This would mean that 22 different numbers of CC subsets would be investigated (from 1 to 22) for this dataset. Even if all 1010 ISBSG projects with good data quality were used as CC projects (Minku and Yao 2017), this would still lead to the investigation of only $C = 101$ different numbers of CC subsets. Therefore, investigating a large portion of (or even all) meaningful values for the number of CC subsets does not demand very large runtime, and does not require advanced algorithms such as the offline evolutionary algorithms mentioned in Section 3.2.

Whenever a new WC training project is received, it is used to update the estimate of the current predictive performance (validation error) of each Dycom instance in an online manner (lines 3 and 4). The current validation error L_i of a given Dycom instance i at a given time step is estimated using (4):

$$L_i = \begin{cases} E_i(y, \hat{y}_i), & \text{if } (\mathbf{x}, y) \text{ is the first WC training project} \\ \alpha L_i + (1-\alpha) \cdot E_i(y, \hat{y}_i), & \text{otherwise} \end{cases} \tag{4}$$

where $E_i(y, \hat{y}_i)$ is the error of Dycom's instance i on a single project $(\mathbf{x}, y)$ for which the estimation was $\hat{y}_i$; and α, $0 \leq \alpha < 1$, is a fading factor (Gama et al. 2009).

This equation enables the tuning procedure to adapt to changes in the data generating process which may result in the best number of CC subsets to change over time. In particular, the fading factor α controls how much importance should be given to the past. Values very close to one lead to more emphasis on the past, resulting in more stable selection of the number of CC subsets, but slower adaptation to changes. Smaller values lead to faster adaptation, but more unstable selection of the number of CC subsets. Following

Shepperd and McDonell's recommendations (Shepperd and McDonell 2012), the absolute error is used as a performance metric for the online supervised hyperparameter tuning procedure in this work, given that it is unbiased towards under or overestimations. Therefore, $E_i(y, \hat{y}_i) = |y - \hat{y}_i|$.

After using a given WC training project for updating the current validation error of each Dycom instance, the procedure uses this WC training project to train each Dycom instance (line 5). Training is performed using the algorithms described in Sections 4 and 5. The use of the WC training projects for training *after* using them to estimate Dycom's predictive performance avoids overly optimistic predictive performance estimations. The Dycom instance with the best estimated predictive performance at a given time step is selected to be used for SEE (line 6) until a new WC training project is received.

6.2 Expectation-Maximisation's Offline Unsupervised Tuning Procedure

Expectation-Maximisation can automatically determine the number of clusters by runing 10-fold cross-validation on the CC training projects being clustered, using the log likelihood as the validation measure. This is done through the procedure shown in Algorithm 4 (Hall et al. 2009). If there are less than 10 projects, the number of folds is set to the number of projects in order to perform cross-validation. In this study, this offline unsupervised tuning procedure is integrated within Dycom as illustrated in Fig. 3.

Algorithm 4 Expectation-maximisation's tuning procedure (Hall et al. 2009).

```
Set the initial number of clusters to $M = 1$.
Randomly split the CC training projects into 10 folds.
Run 10 times 10 fold cross-validation. Specifically, for each fold $f$ of each run, all folds but $f$ are used for clustering using Expectation-Maximisation, whereas $f$ is used for computing the log likelihood.
Average the log likelihood calculated based on the validation folds.
Repeat the process above by incrementing the number of clusters by 1 while improvements in the log likelihood are observed. If no improvements are observed, the procedure stops and retrieves the number of clusters with the maximum log likelihood so far.
```

6.3 X-Means' Offline Unsupervised Tuning Procedure

X-Means contains a built-in mechanism for determining the number of clusters (Pelleg and Moore 2000). The mechanism starts with running the accelerated K-Means with the minimum number of clusters $minC$, which is a pre-defined hyperparameter. After running the accelerated K-Means, the centres of each cluster are split into two children. The two child centroids are moved a distance proportional to the size of the region covered by the cluster in opposite directions, along a randomly chosen vector.

A local accelerated K-Means is then run inside each child cluster, using the two child centroids as the initial centroids. A test is performed on each pair of children to check whether the parent centroid or the new children centroids are better to describe the underlying distribution of the data. This test is used to decide whether to maintain the split of the cluster into two children or not. It is based on the Bayesian Information Criterion, which is defined as the log likelihood of the data given the clusters, minus a value that penalises a larger number of clusters. This penalty is used to avoid overfitting.

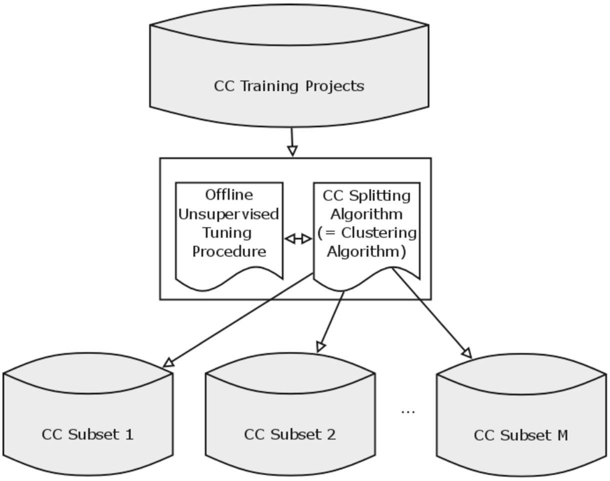

CC Training Projects
Offline Unsupervised Tuning Procedure
CC Splitting Algorithm (= Clustering Algorithm)
CC Subset 1
CC Subset 2
...
CC Subset M

Fig. 3 Offline unsupervised hyperparameter tuning procedure applied to Dycom

If the children clusters describe the data distribution better, the procedure is repeated recursively inside each child cluster, to decide whether to split them further. A maximum number of clusters $maxC$ can also be specified as a hyperparameter. In this study, this offline unsupervised tuning procedure is integrated within Dycom as illustrated in Fig. 3.

7 Datasets

This paper answers the research questions posed in Section 2 by performing a case study based on the ISBSG Repository Release 10 (ISBSG 2011). This repository has been chosen because (1) it contains projects developed by several different companies worldwide, enabling CC projects to be used; (2) the implementation date of projects is available, enabling chronology to be taken into account; and (3) information on which projects belong to a single company for composing a WC dataset was provided to us upon request, enabling software effort estimations to be performed and evaluated for a given company of interest. Datasets without information about the implementation date or where the input attributes were too limited were not used, so that the case study represents real world scenarios.

The ISBSG data were preprocessed as in previous work (Minku and Hou 2017; Minku and Yao 2017; 2014), maintaining only projects with:

- Data and function points quality A (assessed as being sound with nothing being identified that might affect their integrity) or B (appears sound but there are some factors which could affect their integrity).
- Recorded effort that considers only the development team.
- Normalised effort equal to total recorded effort, meaning that the reported effort is the actual effort across the whole life cycle.
- Functional sizing method IFPUG version 4+, or identified as with addendum to existing standards.
- Available implementation date.

The preprocessing resulted in 184 projects from a single-company (WC) and 826 projects from other companies (CC). Four input attributes (development type, language type, development platform and functional size) and one output attribute (software effort in person-hours) were used.

Three different datasets were created, representing different scenarios as follows:

- ISBSG2001 – 224 CC projects implemented from 1993 until the end of 2001, and 69 WC projects implemented from the year 2002 until the end of March 2003 (Minku and Hou 2017).
- ISBSG2000 – 168 CC projects implemented from 1993 until the end of 2000, and 119 WC projects implemented from the year 2001 until the end of March 2003 (Minku and Hou 2017).
- ISBSGAll – 94 CC projects implemented from 1993 until the end of February 1996, and 184 WC projects implemented from March 1996 until the end of March 2003. This scenario uses all of the WC projects provided by ISBSG.

These scenarios represent situations with varying numbers of CC projects and varying lengths of time covered by the WC projects. Different data (e.g., different CC projects) available prior to online learning can greatly influence the predictive performance of online learning approaches (Minku and Yao 2017). In particular, the results of the experiments using this case study show that Dycom's behaviour varies for these datasets, as will be shown in Sections 9 and 10. Therefore, it is useful to investigate these different scenarios, rather than a single scenario. The productivities associated to the CC projects of each scenario are shown in Fig. 4.

8 Experimental Setup

In order to answer the research questions outlined in Section 2, the following approaches were compared:

- **Dycom:** only one at every $P = 10$ projects in the WC projects stream were assumed to have their true effort collected for training. So, Dycom used only 10% of the WC projects for training, as in previous work (Minku and Yao 2014; Minku and Hou 2017). Whenever a new WC training project was made available, the WC model used by Dycom was rebuilt from scratch using all WC training projects so far. This is not a time consuming process because the number of WC projects available for training is small. The following different Dycom approaches were investigated:
  - **Threshold Clustering, K-Means, Hierarchical Clustering, Spectral Clustering:** for simplicitly, we will use the name of the clustering methods to refer to Dycom with a given clustering method and the online supervised tuning procedure explained in Section 6.1. This tuning procedure was used for all clustering methods that do not have a built-in tuning procedure for choosing the number of clusters. For Threshold Clustering, K-Means, Hierarchical Clustering and Spectral Clustering, all numbers of clusters from $M = 1$ to $\lfloor N/10 \rfloor$, where N is the number of CC projects, were investigated. For Spectral Clustering, 19 different α^* values from 0.05 to 0.95 with increments of 0.05 were used to determine the number of clusters.
  - **Expectation-Maximisation:** we will use the name of the clustering method Expectation-Maximisation to refer to Dycom using Expectation-

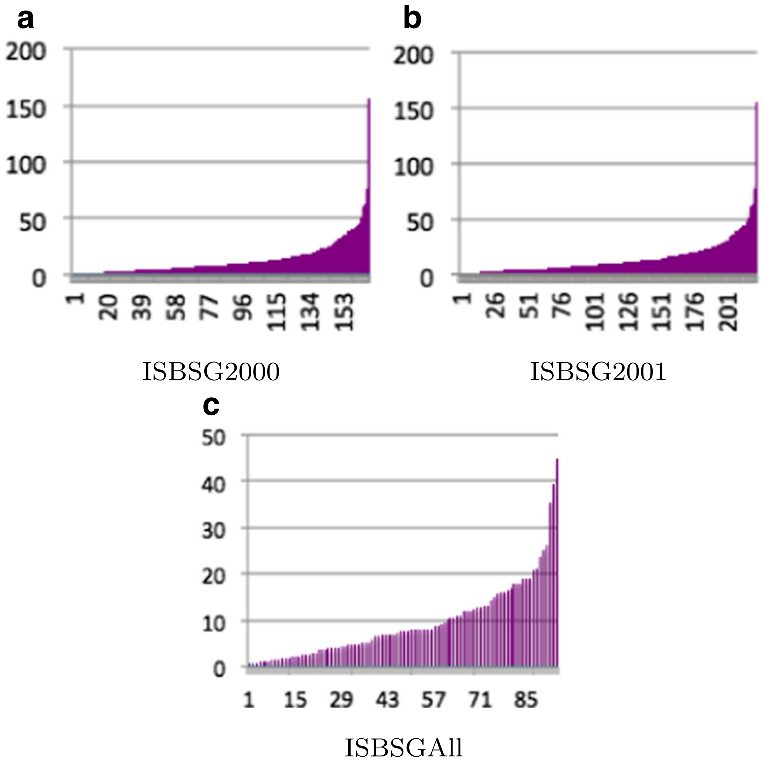


Fig. 4 CC projects' productivities. Y-axis shows effort divided by size. X-axis lists projects sorted from the most productive to the least productive

Maximisation with the offline unsupervised tuning procedure described in Section 6.2. The minimum number of clusters was set as $M = 1$. This procedure does not require a maximum number of clusters to be specified.

  - **X-Means:** we will use the name of the clustering method X-Means to refer to Dycom using X-Means with the offline unsupervised tuning procedure described in Section 6.3. The minimum and maximum number of clusters were set to $minC = 1$ and $maxC = \lfloor N/10 \rfloor$, where N is the number of CC projects.

- **WC:** this is a WC SEE model trained with all WC projects received up to each given time step, i.e., with $P = 1$. This approach is used to check whether Dycom is successful in drastically reducing the number of WC projects required for training. In particular, we can consider Dycom to be successful if it obtains predictive performance similar to or better than WC.
- **WC-P10:** this is a WC SEE model trained with one in every $P = 10$ WC projects. This is used as a control approach to better understand the behaviour of Dycom and of the WC model described above. In particular, if WC-P10 has similar performance to WC, this means that several WC projects are detrimental if used for training. In this case, it would not be necessary to collect the true required effort for a large number of WC projects, even when Dycom is not used.

- **Median-P10:** this is a Median baseline SEE model trained with one in every $P = 10$ WC projects. Median is a typical baseline used in SEE studies. It predicts the median of the efforts of all previously seen WC training projects. The Median baseline was selected instead of the Mean baseline because it can frequently provide better estimates (Minku et al. 2015), as its estimations are not affected by outlier projects. Reasonable SEE approaches should perform better than Median.

Apart from the number of clusters, the clustering hyperparameters were set so as to run the methods described in Section 5. The stopping criteria for K-Means, Expectation-Maximisation and X-Means were set to the same default value of 500 iterations, and Expectation-Maximisation's minimum standard deviation was set to the default value of 1.0E-6. Dycom's hyperparameters β and lr were the same default values of 0.5 and 0.1, respectively, used in Minku and Yao (2014), Minku and Hou (2017).

The use of default values avoids giving an unfair advantage to Dycom over the WC models and Median baselines in the experiments. If Dycom using default hyperparameters performs similar to or better than the WC SEE model, this already represents a success of Dycom. This success could potentially be improved further by fine tuning these hyperparameters. However, as explained in Section 3.2, there are no agreed methods for tuning hyperparameters of online learning algorithms due to the unavailability of pre-existing data to compose a validation set. Future work could potentially investigate the online supervised tuning procedure proposed in Section 6.1 for automatically tuning other hyperparameters than the number of CC subsets. However, given that the default values already lead to successful results (Minku and Yao 2014), it is better to spend the whole budget of hyperparameter value investigations to tune the number of CC subsets, which is known to significantly affect Dycom's predictive performance.

Three different base learning algorithms were used to generate Dycom's, WC's and WC-P10's SEE models:

- **RT:** this base learner was chosen because its locality can potentially help to deal with the heterogeneity among the WC projects. It has been used as a base learner in previous studies with Dycom (Minku and Yao 2014; Minku et al. 2015), and demonstrated to be a competitive algorithm for SEE (Minku and Yao 2013) with respect to several other machine learning algorithms (Multilayer Perceptrons, Radial Basis Function networks, BagRTs, Bagging Ensembles of Multilayer Perceptrons, Bagging ensembles of Radial Basis Function networks, Random ensembles of Multilayer Perceptrons and Negative Correlation Learning ensembles of Multilayer Perceptrons) on a study involving thirteen datasets. In particular, together with BagRT, it was the approach whose Mean Absolute Error (MAE) was less frequently considerably worse than that of the best ranked approach.
- **BagRT:** this base learner was chosen because it can improve predictive performance over RTs (Minku and Yao 2013), even though it produces less readable SEE models. In particular, it was the top ranked approach in terms of MAE in the above mentioned study (Minku and Yao 2013). This rank was similar to that of Bagging ensembles of Multilayer Perceptrons and Bagging ensembles of Radial Basis Function networks (Minku and Yao 2013), but BagRT was less frequently considerably worse than the best ranked approach.
- **LogLR:** this base learner was chosen because it was the top ranked base learner in terms of Spearman's rank correlation in a SEE study involving several machine learning algorithms (Least Squares Support Vector Machines, Case-Based Reasoning, Multilayer Perceptrons, Radial Basis Function networks, M5 model trees, Classification and

> Regression Trees, Multivariate Adaptive Regression Splines, Robust Regression, Ridge Regression, Ordinary Least Squares Regression with Box Cox transformation, and Ordinary Least Squares Regression) on a study involving nine datasets. This rank was similar to that of Ordinary Least Squares with Box Cox transformation, Ridge Regression, M5 model trees, Ordinary Least Squares, Least Squares Support Vector Machines and Classification and Regression Trees.

All models estimate the effort of the development team in person-hours based on development type, language type, development platform and functional size, as per the output and input attributes of the datasets described in Section 7.

LogLR is parameterless, whereas RTs and BagRTs are known not to be much affected by their hyperparameters in the context of online SEE (Song et al. 2013). Therefore, this study does not concentrate on tuning these hyperparameters, and uses the values that most commonly led to the best results in previous work (Minku and Yao 2013) as default values. These are value of 1 for the minimum total weight for the instances in a leaf (for RT and BagRT); value of 0.0001 for the minimum proportion of the variance of all the data that need to be present at a node for splitting to be performed (for RT and BagRT); and value of 50 for the number of RTs (for BagRT).

WEKA's (Hall et al. 2009) implemention was used for all clustering methods and base learners. For Spectral Clustering, the WEKA implementation provided by Luigi Dragone[2] was used. For RT, the REPTree implementation of RTs (Hall et al. 2009) was used. For LogLR, WEKA's implementation of Ridge Regression (Hall et al. 2009) was modified to operate in the logarithmic scale, and the ridge factor was set to a very low value (10^{-8}) to simulate Ordinary Least Squares Regression.

It is worth noting that this is the first time for Dycom to be investigated with other base learners than RTs. Therefore, the results of the experiments are expected to not only lead to answers to the research questions posed in Section 2, but also further insights into the use of Dycom with other base learners.

The predictive performance of all approaches was evaluated based on the time-decayed Absolute Error (AE) and time-decayed AE on the log scale (LogAE). Time-decayed performance evaluation is recommended for online learning studies (Gama et al. 2009), because it enables to track the current predictive performance of a given online learning algorithm over time, reflecting potential changes in the data generating process. Absolute error was adopted because it is unbiased towards under and overestimations (Shepperd and McDonell 2012). Absolute error in the log scale was adopted because it not influenced by project size. These performance measures were computed as follows:

$$AE^t = \begin{cases} |y^t - \hat{y}^t|, & \text{if } t = 1 \\ \alpha AE^{t-1} + (1-\alpha) \cdot |y^t - \hat{y}^t|, & \text{otherwise} \end{cases} \tag{5}$$

$$LogAE^t = \begin{cases} |ln(y^t) - ln(\hat{y}^t)|, \text{if } t = 1 \\ \alpha LogAE^{t-1} + (1-\alpha) \cdot |ln(y^t) - ln(\hat{y}^t)|, & \text{otherwise} \end{cases} \tag{6}$$

where y^t is the actual effort of the WC project received at time t, $\hat{y}^t$ is the effort estimation given by an approach for the example received at time t, and α, $0 \leq \alpha < 1$, is a fading factor similar to that of (4). The fading factor was set to 0.9, both for the performance evaluation of the compared approaches, and for the online supervised hyperparameter tuning procedure.

[2] http://www.luigidragone.com/software/spectral-clusterer-for-weka/

This value was found to provide a good trade-off between smoothness of the performance over time and adaptability to changes. It is the same factor found to be suitable in previous work on online learning (Wang et al. 2015). When summarising the predictive performance across time steps, the median of the AE^t and $LogAE^t$ values across time steps was used. The medians are referred to as $MdAE$ and $MdLogAE$, respectively. Median Absolute Deviation (MdAD) is reported as a measure of the deviations of AE^t and $LogAE^t$ values.

The unit of time-decayed AE is person-hours, whereas the unit of time-decayed LogAE is ln of person-hours. As such, these performance metrics have a minimum value of zero (for ln, the minimum value of zero is guaranteed only for effort values and estimations equal to or larger than 1 person-hour). However, these performance metrics do not have an upper bound. This could make their interpretability difficult. To facilitate the interpretability of the results, Standardised Accuracies (SAs) across time steps with respect to MdAE (SA_{AE}) and MdLogAE (SA_{LogAE}) were also used. They are defined as follows:

$$SA_{AE} = 1 - \frac{MdAE}{MdAE_{median}}, \tag{7}$$

$$SA_{LogAE} = 1 - \frac{MdLogAE}{MdLogAE_{median}}, \tag{8}$$

where $MdAE_{median}$ and $MdLogAE_{median}$ are the $MdAE$ and $MdLogAE$ of the Median-P10 baseline. This metric is similar to that proposed by Shepperd and McDonell (2012), but represents how much better/worse a given approach is than the Median baseline, rather than the Mean. More specifically, positive SA indicates how much better the approach is than the Median baseline. The absolute value of negative SAs indicates how much worse a given approach is than the Median baseline. As explained earlier, the Median baseline tends to perform better than the Mean, leading to a stricter performance comparison.

It is worth noting that all WC projects can be used for evaluating the compared approaches, but only the WC training projects are available for the online hyperparameter tuning procedure to use. This reflects a real world scenario where the true effort is not collected for all WC projects. It is also important to note that no WC training project is used for training before being used for evaluation. This avoids overly optimistic evaluation of predictive performance.

The comparisons among the approaches were supported by the Scott-Knott multiple comparison procedure, as recommended by Mittas and Angelis (2013). This procedure has the advantages of (1) separating different approaches into non-overlapping groups and (2) reducing the number of statistical tests performed, being a powerful test. As suggested by Menzies et al. (2017), Scott-Knott was run with non-parametric bootstrap sampling, making the test non-parametric. In addition, Vargha and Delaney's non-parametric A12 effect size (Vargha and Delaney 2000) was used to further reduce the number of bootstrap tests run by the Scott-Knott procedure, saving running time (Menzies et al. 2017) and ensuring that groups with small differences are merged. Specifically, Scott-Knott only performed statistical tests to check whether groups should be separated if the A12 effect size was medium or large. If A12 effect size was not medium or large, groups were not separated. As suggested by Vargha and Delaney (2000), A12 $\geq$0.56 is considered small, $\geq$0.64 is considered medium and $\geq$0.71 is considered large. The code provided by Tim Menzies[3] was used to

[3] https://github.com/txt/ase16/blob/master/doc/stats.md

Table 1 Median predictive performance across time steps when using RT as base learner

Method	ISBSG2000		ISBSG2001		ISBSGAll	
	MdAE (MdAD)	SA_{AE}	MdAE (MdAD)	SA_{AE}	MdAE (MdAD)	SA_{AE}
Threshold Clustering	2034 (367)	27%	2086 (427)	32%	3139 (608)	17%
K-Means	2112 (464)	25%	2021 (383)	34%	3311 (601)	12%
Hierarchical Clustering	2071 (366)	26%	2195 (208)	28%	3400 (711)	10%
Spectral Clustering	2919 (489)	−04%	2074 (317)	32%	3400 (734)	10%
Expectation-Maximisation	2212 (456)	21%	2364 (426)	23%	4459 (849)	−18%
X-Means	2919 (489)	−04%	2074 (317)	32%	3400 (734)	10%
WC	2869 (744)	−02%	4143 (625)	−35%	2815 (737)	25%
WC-P10	3546 (626)	−26%	3422 (316)	−11%	5211 (1243)	−38%
Median-P10	2804 (426)	00%	3070 (274)	00%	3778 (863)	00%

Method	ISBSG2000		ISBSG2001		ISBSGAll	
	MdLogAE (MdAD)	SA_{LogAE}	MdLogAE (MdAD)	SA_{LogAE}	MdLogAE (MdAD)	SA_{LogAE}
Threshold Clustering	0.77 (0.17)	22%	0.81 (0.14)	27%	0.85 (0.14)	20%
K-Means	0.83 (0.15)	16%	0.83 (0.13)	26%	0.85 (0.14)	20%
Hierarchical Clustering	0.73 (0.11)	26%	0.95 (0.11)	15%	0.85 (0.14)	20%
Spectral Clustering	1.06 (0.13)	−07%	0.88 (0.16)	21%	0.84 (0.17)	21%
Expectation-Maximisation	0.86 (0.14)	12%	0.87 (0.17)	22%	0.91 (0.14)	14%
X-Means	1.06 (0.13)	−07%	0.88 (0.16)	21%	0.84 (0.17)	21%
WC	0.87 (0.10)	11%	1.30 (0.24)	−16%	0.84 (0.12)	21%
WC-P10	1.11 (0.20)	−12%	1.22 (0.13)	−09%	1.14 (0.19)	−07%
Median-P10	0.99 (0.09)	00%	1.12 (0.11)	00%	1.06 (0.19)	00%

Cells in orange (dark grey) highlight negative SA values, indicating poor results in comparison with Median-P10

Table 2 Median predictive performance across time steps when using BagRT as base learner

Method	ISBSG2000		ISBSG2001		ISBSGAll	
	MdAE (MdAD)	SA_{AE}	MdAE (MdAD)	SA_{AE}	MdAE (MdAD)	SA_{AE}
Threshold Clustering	1802 (350)	36%	2107 (432)	31%	3354 (835)	11%
K-Means	2159 (574)	23%	2109 (448)	31%	3754 (1135)	01%
Hierarchical Clustering	2283 (406)	19%	2348 (317)	24%	4117 (988)	−09%
Spectral Clustering	2880 (607)	−03%	2218 (415)	28%	4253 (806)	−13%
Expectation-Maximisation	2217 (456)	21%	2266 (546)	26%	2823 (605)	25 %
X-Means	2880 (607)	−03%	2218 (415)	28%	4253 (806)	−13%
WC	2691 (834)	04%	3668 (494)	−19%	2417 (650)	36%
WC-P10	3364 (579)	−20%	3397 (414)	−11%	4906 (1419)	−30%
Median-P10	2804 (426)	00%	3070 (274)	00%	3778 (863)	00%

Method	ISBSG2000		ISBSG2001		ISBSGAll	
	MdLogAE (MdAD)	SA_{LogAE}	MdLogAE (MdAD)	SA_{LogAE}	MdLogAE (MdAD)	SA_{LogAE}
Threshold Clustering	0.79 (0.16)	20%	0.84 (0.16)	25%	0.85 (0.10)	20%
K-Means	0.77 (0.15)	22%	0.83 (0.16)	26%	0.89 (0.14)	16%
Hierarchical Clustering	0.80 (0.10)	19%	0.98 (0.12)	12%	0.93 (0.14)	12%
Spectral Clustering	1.01 (0.15)	−02%	0.90 (0.17)	19%	0.91 (0.18)	14%
Expectation-Maximisation	0.79 (0.09)	20%	0.83 (0.18)	25%	0.79 (0.11)	25%
X-Means	1.01 (0.15)	−02%	0.90 (0.17)	19%	0.91 (0.18)	14%
WC	0.78 (0.08)	21%	1.19 (0.23)	−07%	0.75 (0.11)	30%
WC-P10	1.11 (0.16)	−12%	1.21 (0.16)	−08%	1.16 (0.16)	−09%
Median-P10	0.99 (0.09)	00%	1.12 (0.11)	00%	1.06 (0.19)	00%

Cells in orange (dark grey) highlight negative SA values, indicating poor results in comparison with Median-P10

Table 3 Median predictive performance across time steps when using LogLR as base learner

Method	ISBSG2000 MdAE (MdAD)	SA_{AE}	ISBSG2001 MdAE (MdAD)	SA_{AE}	ISBSGAll MdAE (MdAD)	SA_{AE}
Threshold Clustering	2208 (576)	21%	2319 (750)	24%	2451 (573)	35%
K-Means	2119 (526)	24%	2319 (760)	24%	2422 (561)	36%
Hierarchical Clustering	2162 (569)	23%	2194 (508)	29%	2461 (543)	35%
Spectral Clustering	2486 (734)	11%	2640 (866)	14%	2728 (756)	28%
Expectation-Maximisation	2345 (618)	16%	2539 (858)	17%	2561 (589)	32%
X-Means	2486 (734)	11%	2640 (866)	14%	2728 (756)	28%
WC	2267 (432)	19%	2591 (557)	16%	2224 (504)	41%
WC-P10	3369 (1139)	−20%	3968 (1269)	−29%	2700 (909)	29%
Median-P10	2804 (426)	00%	3070 (274)	00%	3778 (863)	00%

Method	ISBSG2000 MdLogAE (MdAD)	SA_{LogAE}	ISBSG2001 MdLogAE (MdAD)	SA_{LogAE}	ISBSGAll MdLogAE (MdAD)	SA_{LogAE}
Threshold Clustering	0.83 (0.14)	16%	0.85 (0.23)	24%	0.74 (0.10)	30%
K-Means	0.82 (0.13)	17%	0.85 (0.23)	24%	0.74 (0.10)	31%
Hierarchical Clustering	0.77 (0.10)	22%	0.94 (0.11)	16%	0.74 (0.10)	30%
Spectral Clustering	0.88 (0.16)	11%	0.89 (0.18)	20%	0.78 (0.08)	27%
Expectation-Maximisation	0.82 (0.15)	17%	0.86 (0.25)	23%	0.73 (0.10)	31%
X-Means	0.88 (0.16)	11%	0.89 (0.18)	20%	0.78 (0.08)	27%
WC	0.80 (0.11)	19%	0.95 (0.16)	15%	0.75 (0.11)	29%
WC-P10	1.09 (0.24)	−10%	1.26 (0.24)	−13%	0.82 (0.10)	23%
Median-P10	0.99 (0.09)	00%	1.12 (0.11)	00%	1.06 (0.19)	00%

Cells in orange (dark grey) highlight negative SA values, indicating poor results in comparison with Median-P10

run the Scott-Knott procedure. Tim Menzies also shared the latex commands to generate quartile plots in a concise and informative way to analyse the results.

It is worth mentioning that Tantithamthavorn et al. (2017, 2018) also recommended to use effect size and strategies to avoid problems with violations of normality when using Scott-Knott. The differences are that their proposed variation of Scott-Knott is based on applying log transformation to avoid violations of normality and Cohen's effect size (Cohen 1992) as the (sole) criterion for deciding whether to merge different groups. mIn addition, they enable groups to be split based on small effect sizes, whereas the procedure explained in the previous paragraph ensures that groups can only be split if medium or large effect sizes exist between them. Therefore, the results of the application of Tantithamthavorn et al. (2017, 2018)'s procedure and the procedure explained in the previous paragraph are unlikely to be the same. Tantithamthavorn et al. (2017, 2018)'s procedure would result in some algorithms or configurations being considered significantly different despite their differences having only a small effect size, whereas the procedure described in the previous paragraph would not. As medium or large effect sizes are more likely to be meaningful in practice, this paper adopted the procedure described in the previous paragraph.

RTs, LogLRs, Threshold Clustering, Hierarchical Clustering and Spectral Clustering are deterministic, whereas BagRTs, K-Means, Expectation-Maximisation and X-Means are non-deterministic. Experiments involving any non-deterministic algorithm were run 30 times. The median of the results obtained across these 30 runs was used for evaluation purposes.

9 RQ1—Improving Predictive Performance When a Limited Number of WC Training Projects is Available

This section presents the analysis done to answer RQ1: Which clustering and tuning approach is the most successful in improving Dycom's predictive performance in comparison to other approaches that also have access to a limited number of WC training projects?

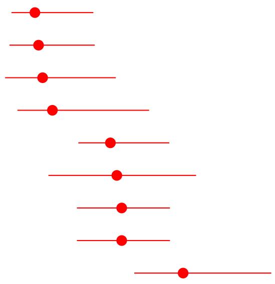
Tables 1, 2 and 3 show the median predictive performance across time steps when using RT, BagRT and LogLR as base learners, respectively. Among all Dycom approaches, Threshold Clustering and K-Means are the only two that never led to negative SA values. Negative SAs represent predictive performance similar to or worse than Median-P10, meaning that it is not worth adopting the approach. Therefore, all Dycom approaches but Threshold Clustering and K-Means can be risky to adopt, as they cannot always avoid poor CC splits.

However, to check how successful Threshold Clustering and K-Means are in improving predictive performance when there is a limited number of WC training projects, we still need to further analyse how these Dycom approaches perform in comparison with Median-P10 and WC-P10. In particular, if the positive SA values obtained by these Dycom approaches are still linked to similar MdAE and MdLogAE to Median-P10 or WC-P10, they are still not successful in improving predictive performance when there is a limited number of WC training projects. Therefore, to investigate this further, Tables 4, 5 and 6 show the results of the Scott-Knott procedure to compare all approaches when using RT, BagRT and LogLR as base learners, respectively. As each table contains the results for all approaches using the same base learner, we can separate the effect of the base learner from the effect of the clustering method and its associated tuning procedure, facilitating the analysis to answer RQ1.

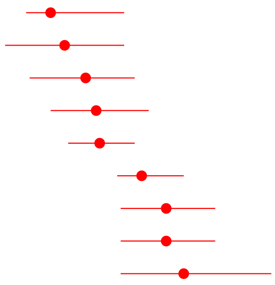
Table 4 Results of Scott-Knott procedure when using RT as base learner

ISBSG2000

Rank	Method	MdAE	IQR
1	Threshold Clustering	2034	835
1	Hierarchical Clustering	2071	871
1	K-Means	2112	1130
2	Expectation-Maximisation	2212	1344
3	Median-P10	2804	928
3	WC	2869	1506
3	X-Means	2919	949
3	Spectral Clustering	2919	949
4	WC-P10	3546	1398

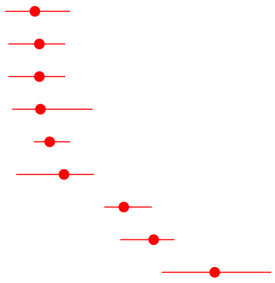


Rank	Method	MdLogAE	IQR
1	Hierarchical Clustering	0.73	0.28
1	Threshold Clustering	0.77	0.34
1	K-Means	0.83	0.30
2	Expectation-Maximisation	0.86	0.28
2	WC	0.87	0.19
3	Median-P10	0.99	0.19
4	X-Means	1.06	0.27
4	Spectral Clustering	1.06	0.27
5	WC-P10	1.11	0.43



ISBSG2001

Rank	Method	MdAE	IQR
1	K-Means	2021	767
1	X-Means	2074	670
1	Spectral Clustering	2074	670
2	Threshold Clustering	2086	949
2	Hierarchical Clustering	2195	431
2	Expectation-Maximisation	2364	916
3	Median-P10	3070	559
4	WC-P10	3422	638
5	WC	4143	1293



Rank	Method	MdLogAE	IQR
1	Threshold Clustering	0.81	0.28
1	K-Means	0.83	0.27
2	Expectation-Maximisation	0.87	0.34
2	X-Means	0.88	0.35
2	Spectral Clustering	0.88	0.35
3	Hierarchical Clustering	0.95	0.21
4	Median-P10	1.12	0.26
4	WC-P10	1.22	0.27
5	WC	1.30	0.48

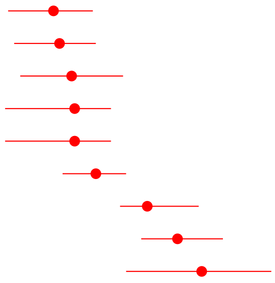
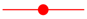
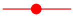
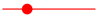
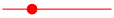
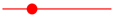
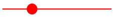
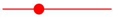
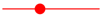
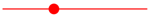
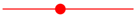
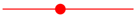
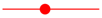
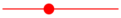
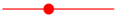
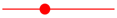
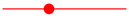
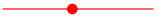
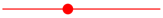


Table 4 (continued)

ISBSGAll

Rank	Method	MdAE	IQR	
1	WC	2815	1481	
2	Threshold Clustering	3139	1234	
3	K-Means	3311	1678	
3	X-Means	3400	1969	
3	Spectral Clustering	3400	1969	
3	Hierarchical Clustering	3400	1970	
3	Median-P10	3778	1969	
4	Expectation-Maximisation	4459	1755	
5	WC-P10	5211	2617	

Rank	Method	MdLogAE	IQR	
1	X-Means	0.84	0.34	
1	Spectral Clustering	0.84	0.34	
1	WC	0.84	0.25	
1	Hierarchical Clustering	0.85	0.30	
1	Threshold Clustering	0.85	0.29	
1	K-Means	0.85	0.29	
2	Expectation-Maximisation	0.91	0.32	
3	Median-P10	1.06	0.39	
4	WC-P10	1.14	0.41	

Rank is the rank retrieved by Scott-Knott, IQR is the Inter-Quartile Range (75–25th percentiles). The IQR range is shown in the right column with black dot at the median

According to the Scott-Knott procedure, Threshold Clustering always outperformed Median-P10. As the Scott-Knott procedure already takes the effect size A12 into account, this means that the differences between Threshold Clustering and Median-P10 have medium or large effect size. From Tables 1 to 3, Threshold Clustering's SA_{AE} and SA_{LogAE} varied from 11% to 36% and from 16% to 30%, respectively. This means that Threshold Clustering performs from 11% to 36% better than Median-P10 in terms of MdAE, and from 16% to 30% better than Median-P10 in terms of MdLogAE. The magnitudes of such improvements could be considered as large, being potentially practically significant.

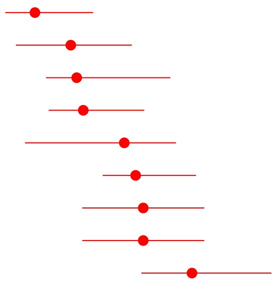
According to the Scott-Knott procedure, K-Means did not outperform Median-P10 in terms of MdAE for ISBSGAll when using RT as the base learner (Table 4). Therefore, the only clustering method that always led Dycom to achieve better results than Median-P10 was Threshold Clustering with the proposed online supervised tuning procedure. This Dycom approach can thus be considered more successful than the other Dycom approaches in improving predictive performance when there are few WC training projects.

According to the Scott-Knott procedure, Threshold Clustering and K-Means performed similarly in 7 out of 9 cases in terms of MdAE, and in 8 out of 9 cases in terms of MdLogAE. For the 3 cases where there were significant differences in terms of MdAE, 2 were in favour of Threshold Clustering and 1 in favour of K-Means. The difference in SA_{AE} among these approaches was up to 13% when Threshold Clustering won, but only 2% when

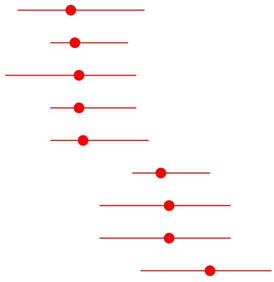
Table 5 Results of Scott-Knott procedure when using BagRT as base learner

ISBSG2000

Rank	Method	MdAE	IQR
1	Threshold Clustering	1802	874
2	K-Means	2159	1154
3	Expectation-Maximisation	2217	1237
3	Hierarchical Clustering	2283	952
3	WC	2691	1503
4	Median-P10	2804	928
4	X-Means	2880	1215
4	Spectral Clustering	2880	1215
5	WC-P10	3364	1294

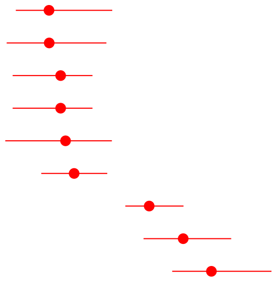


Rank	Method	MdLogAE	IQR
1	K-Means	0.77	0.31
1	WC	0.78	0.19
1	Threshold Clustering	0.79	0.32
1	Expectation-Maximisation	0.79	0.21
1	Hierarchical Clustering	0.80	0.24
2	Median-P10	0.99	0.19
2	X-Means	1.01	0.32
2	Spectral Clustering	1.01	0.32
3	WC-P10	1.11	0.32



ISBSG2001

Rank	Method	MdAE	IQR
1	Threshold Clustering	2107	927
1	K-Means	2109	959
1	X-Means	2218	767
1	Spectral Clustering	2218	767
1	Expectation-Maximisation	2266	1025
2	Hierarchical Clustering	2348	634
3	Median-P10	3070	559
4	WC-P10	3397	842
4	WC	3668	954



Rank	Method	MdLogAE	IQR
1	K-Means	0.83	0.33
1	Expectation-Maximisation	0.83	0.35
1	Threshold Clustering	0.84	0.31
2	X-Means	0.90	0.34
2	Spectral Clustering	0.90	0.34
3	Hierarchical Clustering	0.98	0.26
4	Median-P10	1.12	0.26
4	WC	1.19	0.46
4	WC-P10	1.21	0.32

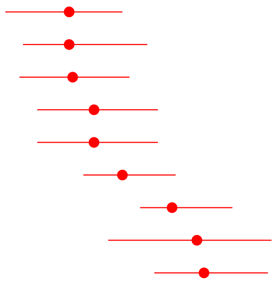
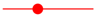
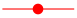
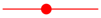
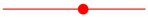
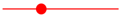
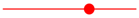
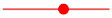
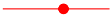
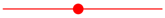
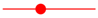
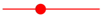
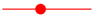
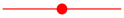
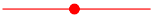
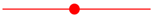
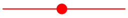
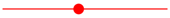
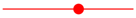


Table 5 (continued)

ISBSGAll

Rank	Method	MdAE	IQR	
1	WC	2417	1531	
2	Expectation-Maximisation	2823	1219	
3	Threshold Clustering	3354	1623	
3	K-Means	3754	2414	
4	Median-P10	3778	1969	
4	Hierarchical Clustering	4117	2268	
4	X-Means	4253	1804	
4	Spectral Clustering	4253	1804	
5	WC-P10	4906	2711	

Rank	Method	MdLogAE	IQR	
1	WC	0.75	0.22	
2	Expectation-Maximisation	0.79	0.23	
3	Threshold Clustering	0.85	0.21	
4	K-Means	0.89	0.28	
4	X-Means	0.91	0.35	
4	Spectral Clustering	0.91	0.35	
4	Hierarchical Clustering	0.93	0.29	
5	Median-P10	1.06	0.39	
5	WC-P10	1.16	0.31	

Rank is the rank retrieved by Scott-Knott, IQR is the Inter-Quartile Range (75 − 25th percentiles). The IQR range is shown in the right column with black dot at the median

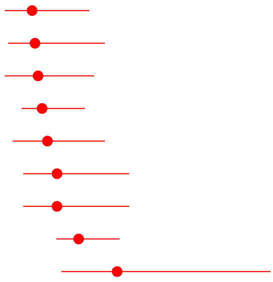
K-Means won. The difference in SA_{LogAE} among these approaches was 4% in favour of Threshold Clustering. This shows that these two approaches frequently performed similarly. However, the magnitude of the difference of 13% in favour of Threshold Clustering can be considered fairly large, being potentially practically significant. In practice, even if the differences in predictive performance between K-Means and Threshold Clustering are small or non-existent, there would be no reason for practitioners to adopt K-Means instead of the much simpler Threshold Clustering method.

The Scott-Knott procedure also shows that both Threshold Clustering and K-Means always managed to obtain better predictive performance than WC-P10. In fact, WC-P10 performed very poorly, frequently even worse than Median-P10. As shown in Tables 1 to 3, its SA_{AE} varied from −38% to 29%, having always been smaller than -10% except for one case. Its SA_{LogAE} varied from −13% to 23%, having always been negative except for one case. These values are much lower than those obtained by Threshold Clustering and K-Means. In particular, the differences in SA_{AE} and SA_{LogAE} between Threshold Clustering and WC-P10 were of up to 56% and 37%, respectively, and were typically higher than 40% and 25%, respectively. Such large values demonstrate that the practical significance of the difference in predictive performance is likely to be very high. As Threshold Clustering and K-Means use the same number of WC training projects as WC-P10, their improved predictive performance is due to their ability to benefit from CC projects.

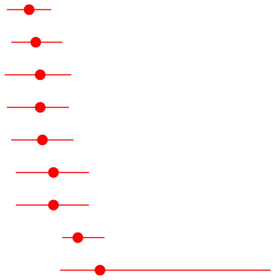
Table 6 Results of Scott-Knott procedure when using LogLR as base learner

ISBSG2000

Rank	Method	MdAE	IQR
1	K-Means	2119	1240
1	Hierarchical Clustering	2162	1421
1	Threshold Clustering	2208	1314
1	WC	2267	930
1	Expectation-Maximisation	2345	1355
2	X-Means	2486	1555
2	Spectral Clustering	2486	1555
3	Median-P10	2804	928
4	WC-P10	3369	3072

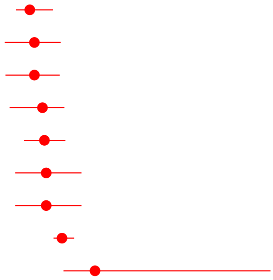


Rank	Method	MdLogAE	IQR
1	Hierarchical Clustering	0.77	0.20
1	WC	0.80	0.23
1	Expectation-Maximisation	0.82	0.30
1	K-Means	0.82	0.28
1	Threshold Clustering	0.83	0.28
2	X-Means	0.88	0.33
2	Spectral Clustering	0.88	0.33
3	Median-P10	0.99	0.19
4	WC-P10	1.09	0.95



ISBSG2001

Rank	Method	MdAE	IQR
1	Hierarchical Clustering	2194	1005
1	K-Means	2319	1523
1	Threshold Clustering	2319	1479
1	Expectation-Maximisation	2539	1489
1	WC	2591	1127
2	X-Means	2640	1805
2	Spectral Clustering	2640	1805
2	Median-P10	3070	559
3	WC-P10	3968	5634



Rank	Method	MdLogAE	IQR
1	Threshold Clustering	0.85	0.39
1	K-Means	0.85	0.38
1	Expectation-Maximisation	0.86	0.40
1	X-Means	0.89	0.37
1	Spectral Clustering	0.89	0.37
1	Hierarchical Clustering	0.94	0.24
1	WC	0.95	0.31
2	Median-P10	1.12	0.26
3	WC-P10	1.26	0.56

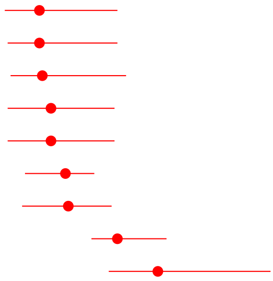
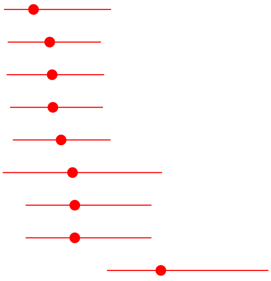


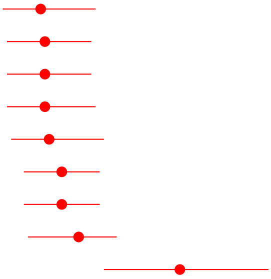
Table 6 (continued)

ISBSGAll

Rank	Method	MdAE	IQR
1	WC	2224	1306
1	K-Means	2422	1138
1	Threshold Clustering	2451	1189
1	Hierarchical Clustering	2461	1134
1	Expectation-Maximisation	2561	1193
2	WC-P10	2700	1945
2	X-Means	2728	1535
2	Spectral Clustering	2728	1535
3	Median-P10	3778	1969



Rank	Method	MdLogAE	IQR
1	Expectation-Maximisation	0.73	0.22
1	K-Means	0.74	0.20
1	Threshold Clustering	0.74	0.20
1	Hierarchical Clustering	0.74	0.21
1	WC	0.75	0.22
2	X-Means	0.78	0.18
2	Spectral Clustering	0.78	0.18
2	WC-P10	0.82	0.21
3	Median-P10	1.06	0.39



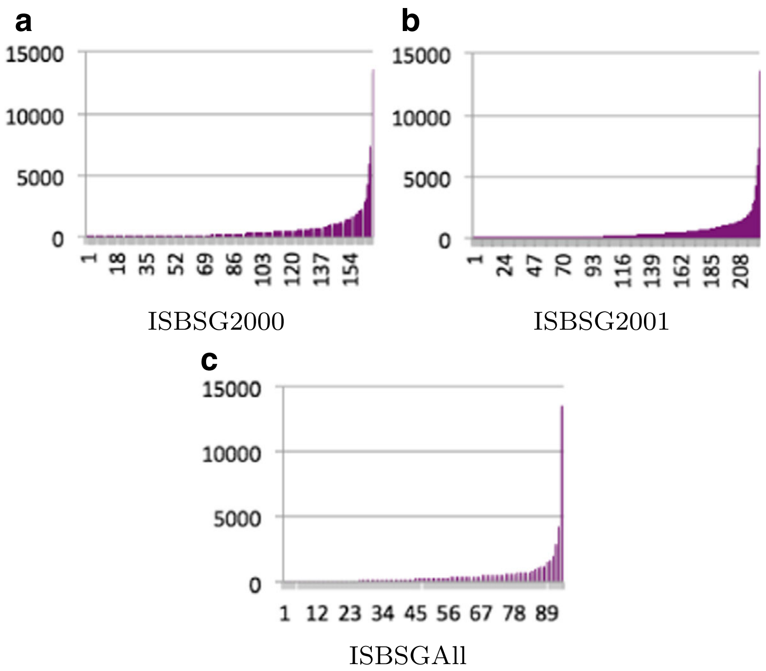
Rank is the rank retrieved by Scott-Knott, IQR is the Inter-Quartile Range (75 − 25th percentiles). The IQR range is shown in the right column with black dot at the median

RQ1: In summary, Threshold Clustering (using the proposed online supervised tuning procedure) was the most successful Dycom approach in improving predictive performance in comparison to other approaches with access to a limited number of WC training projects. It always managed to improve the MdAE and MdLogAE of WC-P10 (a WC model trained on few WC projects), with differences in SA_{AE} and SA_{LogAE} typically higher than 40% and 25%, respectively. It was the only approach to always outperform the Median-P10 baseline in terms of these performance metrics.

10 RQ2—Why Threshold Clustering Worked Well?

This section answers RQ2: Why is Threshold Clustering the most successful Dycom approach in improving predictive performance in comparison to other approaches that also have access to a limited number of WC training projects?

The main reason for this approach to have been competitive is that the functional size of CC training projects was exponentially distributed and not really clustered. This is illustrated in Fig. 5. As a result, more complex clustering algorithms than Threshold were either unable to find more than one cluster, or created skewed clusters, where some clusters contained very few projects. A single cluster mixes very heterogeneous projects, leading to



ISBSG2000

ISBSG2001

ISBSGAll

Fig. 5 CC projects' functional sizes. Y-axis shows functional size. X-axis lists projects sorted from the smallest to the largest

unsuitable CC models. Small clusters have too few training examples, leading to weak CC models, which are again not useful. They were therefore inadequate for use with Dycom with these CC training projects. Threshold Clustering ensures that CC subsets have a balanced number of CC training projects, being expected to perform better for CC training projects distributed in this way. The proposed online supervised hyperparameter tuning procedure then helped to use Threshold Clustering without having to manually tune its number of clusters (CC subsets).

Table 7 gives an example of the sizes of the clusters obtained by different clustering algorithms. Threshold Clustering, by definition, always produced clusters of similar size. Spectral Clustering and X-Means failed to find more than one cluster, except for X-Means for ISBSGAll. X-Means for ISBSGAll, Expectation-Maximisation, Hierarchical Clustering and K-Means produced skewed clusters, with some clusters containing even a single CC training project. K-Means' clusters were skewed to a much lesser extent than the other methods. This is in line with the fact that K-Means was the most competitive clustering algorithm against Threshold Clustering.

RQ2: In summary, the main reason for Threshold Clustering to have performed better than other clustering methods was because the CC training projects were exponentially distributed and not really clustered.

Table 7 Example of cluster sizes produced by different clustering algorithms

	ISBSG2000	ISBSG2001	ISBSGAll
Threshold Clustering	56	75	276
	56	75	276
	56	74	275
K-Means	21	27	99
	58	73	254
	89	124	473
Hierarchical Clustering	1	1	1
	20	12	5
	147	211	820
Spectral Clustering	168	224	826
Expectation-Maximisation	4	10	13
	44	73	81
	121	141	196
			224
			312
X-Means			12
	168	224	814

For the algorithms that require the number of clusters to be pre-defined (Threshold Clustering, K-Means and Hierarchical Clustering), the number of clusters presented in the table is 3. For Spectral Clustering, the number of clusters was always the same independent of its hyperparameter α used. For the others (Expectation-Maximisation and X-Means), the number of clusters most commonly determined by the algorithm for each given dataset is presented. For the stochastic algorithms, the rounded average of the cluster sizes across 30 runs is presented

11 RQ3—Predictive Performance in Comparison to Approaches that Use More WC Training Projects

From Section 9, we can see that Threshold Clustering was in general the most successful Dycom approach in improving predictive performance in comparison to other approaches that also have access to a limited number of WC training projects. The current section presents the analysis done to answer to RQ3: Is this approach to Dycom always successful in reducing the number of WC projects required for training while maintaining predictive performance in comparison to WC models trained on more WC projects?

When comparing Threshold Clustering against WC, the Scott-Knott results (Tables 4 to 6) show that Threshold Clustering was ranked similar or better than WC in most cases (15 out of 18). Therefore, Threshold Clustering was usually successful in maintaining (or even improving) WC's predictive performance while requiring 10 times less WC training projects.

The 3 out of 18 cases where Threshold Clustering did not manage to achieve that were in terms of MdAE when using RT and BagRT as base learners for ISBSGAll (5th subtables within Tables 4 and 5), and in terms of MdLogAE when using BagRT as base learner for ISBSGAll (6th subtable within Table 5). In these cases, the differences in SA_{AE} between Threshold Clustering and WC were 8% and 25%, respectively, and the difference in SA_{LogAE} was 10% (Tables 1 and 2). These are differences of considerable size, meaning that the CC splits obtained by Threshold Clustering were not always successful in maintaining WC's predictive performance while using 10 times less WC training projects.

Additional experiments giving Threshold Clustering access to $P = 5$ rather than 10 times less WC training projects than WC show that $P = 5$ enabled Threshold Clustering to achieve similar performance in terms of MdAE when using RT as base learner for ISBSGAll. The results of the Scott-Knott procedure including Threshold Clustering with $P = 5$ are shown in Table 8. However, when using BagRT, this number of WC training projects was still not enough to enable Threshold Clustering to achieve similar predictive performance to WC. Therefore, even though Threshold Clustering was successful in improving predictive performance with respect to WC-P10, it was unable to maintain the predictive performance of WC in this case.

> *RQ3: In summary, Threshold Clustering managed to produce CC splits that enabled Dycom to maintain or even improve the MdAE and MdLogAE of a corresponding WC model while using 10 times less WC training projects in most (15 out of 18), albeit not all cases.*

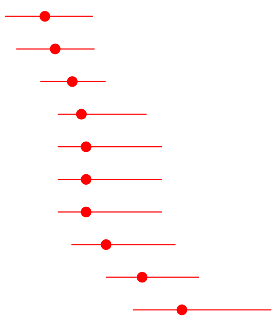
12 RQ4—Why Threshold Clustering Sometimes Failed

As Threshold Clustering sometimes did not manage to reduce the number of WC training projects so drastically while maintaining WC's predictive performance, this section investigates RQ4: Why does it sometimes fail to maintain predictive performance with respect to WC models trained on more WC projects? Can that give us any insights into how to use it?

Table 8 Results of Scott-Knott procedure when using RT as base learner, including threshold clustering with $P = 5$

ISBSGAll

Rank	Method	MdAE	IQR
1	Threshold Clustering-P5	2621	1662
1	WC	2815	1481
2	Threshold Clustering	3139	1234
3	K-Means	3311	1678
3	X-Means	3400	1969
3	Spectral Clustering	3400	1969
3	Hierarchical Clustering	3400	1970
3	Median-P10	3778	1969
4	Expectation-Maximisation	4459	1755
5	WC-P10	5211	2617



Rank is the rank retrieved by Scott-Knott, IQR is the Inter-Quartile Range (75 − 25th percentiles). The IQR range is shown in the right column with black dot at the median

According to Tables 4, 5 and 6, we can see that all cases for which Threshold Clustering did not maintain WC's predictive performance while using 10 times less WC training projects involved ISBSGAll, and either RT or BagRT. To understand Threshold Clustering's behaviour on this dataset in more detail, Fig. 6 shows its time-decayed AE over time when using RT, BagRT and LogLR as the base learners. Time decayed AE is also shown for WC and the Dycom instances investigated by the proposed online supervised tuning procedure. For simplicity, we will refer to time-decayed AEs as simply AEs in this section.

When reading these plots, it is not necessary to distinguish between individual lines representing different Dycom instances selected for use, or between individual lines representing different Dycom instances not selected for use by the online supervised tuning procedure. Instead, these lines are shown to give an idea of how much better or worse the AEs of such Dycom instances is in comparison to those of WC and Threshold Clustering.

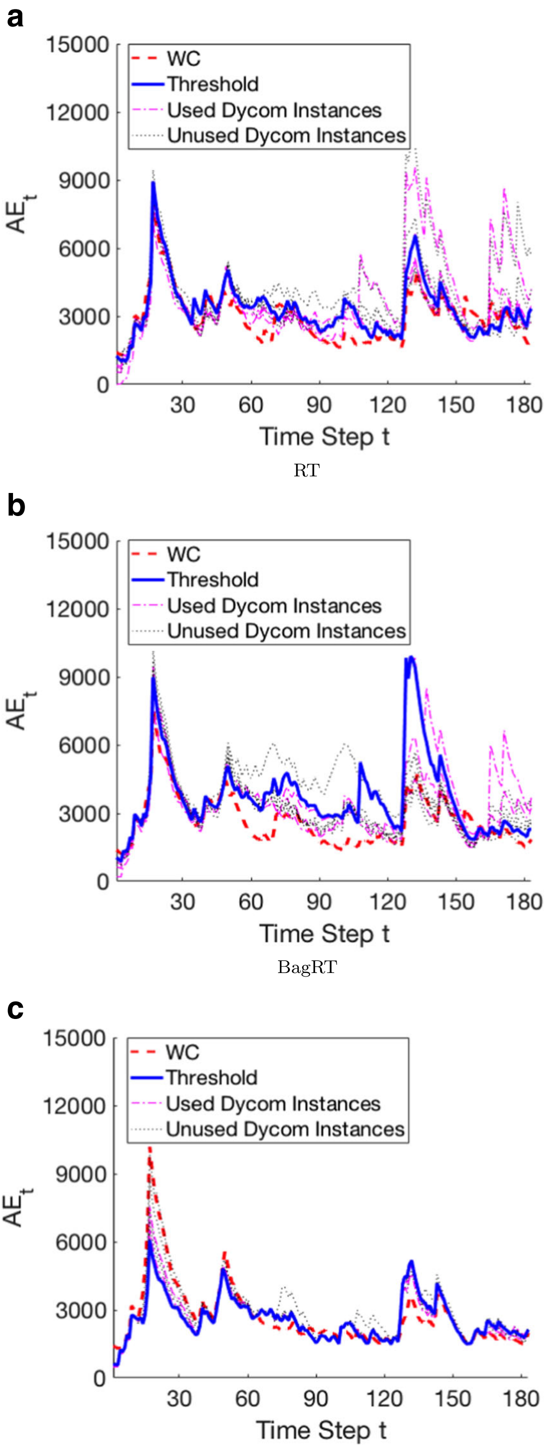
The AEs of some of the Dycom instances were larger than WC's from around time steps 65 to 180 when using RTs (Fig. 6a), from around time steps 60 to 120 and 150 to 180 when using BagRT (Fig. 6b), and around time step 20 when using LogLR (Fig. 6c). The AEs obtained by Threshold Clustering were smaller than those obtained by some Dycom instances during these time periods. Therefore, the online supervised tuning procedure managed to avoid the poor predictive performance of some Dycom instances. However, when using RTs and BagRT, WC still performed better than Threshold Clustering, specially during the middle range of the time period analysed (Fig. 6a and b), as shown by its lower AEs.

Part of the reason for Threshold Clustering's poorer AEs when using RTs and BagRTs is that, even though the tuning procedure was successful in avoiding poor Dycom instances, it did not always select the best possible Dycom instance. For example, the AEs of some of the unused Dycom instances were closer to those obtained by WC during the middle range of the time period analysed in Fig. 6b. Had the online supervised tuning procedure selected those Dycom instances to be used during those time periods, Dycom would have obtained AEs closer to WC's.

As explained in Section 9, the use of $P = 5$ was enough to make Threshold Clustering successful in maintaining WC's AE when using RTs (Table 8). To investigate if $P = 5$ led to better choices of Dycom instances, Fig. 7a shows Threshold Clustering's AEs when using RTs with $P = 5$. We can see an improvement in the AEs obtained by Threshold Clustering in comparison to when $P = 10$ was used (Fig. 6a). In particular, around time steps 20, 60, 100 and 130, Threshold Clustering's AE when using $P = 5$ was more similar to the best AEs among the Dycom instances than when using $P = 10$. This indicates that the tuning procedure was more successful at selecting the best Dycom instances when using $P = 5$ in this scenario.

To complement this analysis, Fig. 8 shows the number of CC subsets selected by the tuning procedure when using Threshold Clustering with RTs and $P = 10$ and $P = 5$. As we can see, $P = 5$ led to more changes in the number of CC subsets (in total 6 different values were used instead of 4). For instance, the tuning procedure chose 4 CC subsets from time steps 16 to 20 when using $P = 5$, but 1 CC subset when using $P = 10$. At around time step 100, $P = 5$ led to the choice of 5 CC subsets, whereas $P = 10$ led to the choice of 6 CC subsets. Around time 130, $P = 5$ again led to the use of 5 CC subsets, whereas $P = 10$ led to the choice of 7 CC subsets. These different choices correspond to time periods when $P = 5$ led Threshold Clustering to better AEs than when using $P = 10$ (Figs. 6a and 7a). This analysis indicates that more frequent collection of WC training projects enables the tuning procedure to better keep track of the best number of CC subsets to be used.

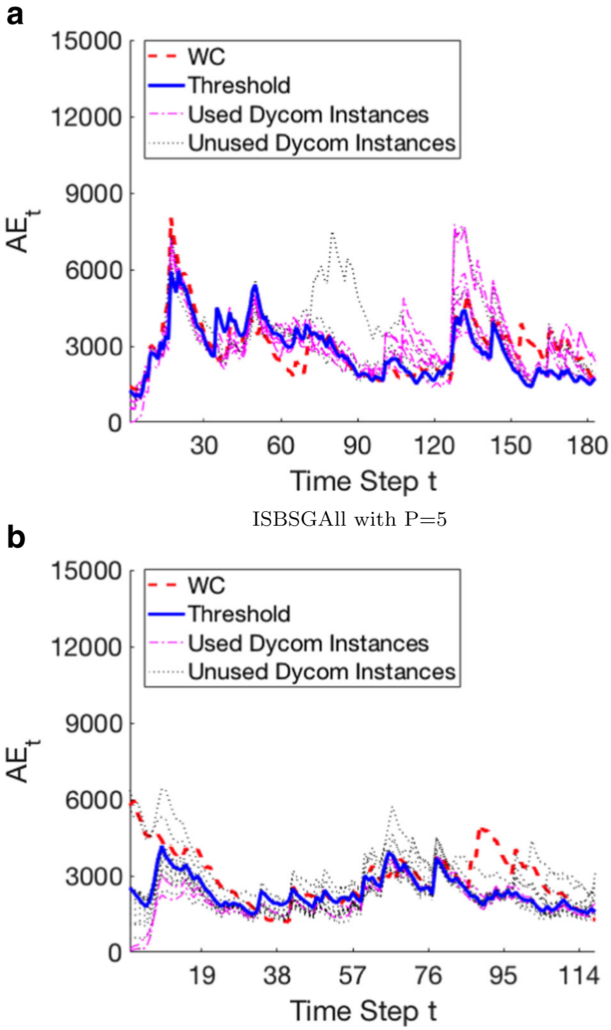
Fig. 6 Time-decayed AE over time for ISBSGAll. Dycom instances that were selected for use at any given point in time by the online supervised tuning procedure are represented using lines of the same style. The same is valid for Dycom instances that were never selected for use at any point in time



RT

BagRT

LogLR



ISBSGAll with P=5

ISBSG2000 with P=10

Fig. 7 Time-decayed AE over time for When Using RT. Dycom instances that were selected for use at any given point in time by the online supervised tuning procedure are represented using lines of the same style. The same is valid for Dycom instances that were never selected for use at any point in time

The investigation above partly reveals the reasons why Threshold Clustering sometimes failed. However, an investigation of the reason why a larger number of WC training projects was necessary could potentially provide more insights into Threshold Clustering's behaviour and how to use it. From Fig. 6, we can see that the AEs obtained by the Dycom instances varied much more for RT and BagRT (Fig. 6a and b) than for LogLR (Fig. 6c). This could cause some of the selected Dycom instances with good AEs in the beginning of the period analysed to perform poorly in the end of this period. When using LogLR, the AEs of different Dycom instances varied much less throughout time. As a result, instances that performed well in the beginning of the period analysed are more likely to still perform well

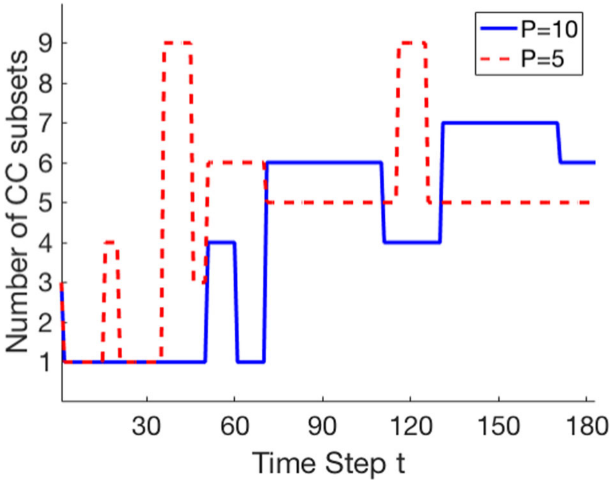


Fig. 8 Number of CC subsets selected by Threshold Clustering with RTs for ISBSGAll when using $P = 10$ and $P = 5$

by the end of the period analysed. This makes it easier for the tuning procedure to avoid poor choices of Dycom instances, requiring less WC training projects.

Figure 7b shows the AEs obtained by Threshold Clustering with RTs on a dataset (ISBSG2000) for which this approach was successful in maintaining WC's AE with $P = 10$. We can see that the variations in the AEs of the Dycom instances were larger than those obtained with LogLR in Fig. 6c, but still smaller than those obtained when using RT in Fig. 6a. This again makes it easier for the tuning procedure to avoid poor choices of Dycom instances, requiring less frequent collection of WC training projects.

Therefore, when the predictive performance of Dycom's instances varies more, a more frequent collection of WC training projects is necessary to enable selecting good Dycom instances. Such larger variance could possibly be observed during runtime based on the WC training projects, and used to decide whether to collect more or less WC training projects. However, a more detailed understanding of the reasons why this variance was larger for ISBSGAll when using RTs and BagRTs could potentially support early decisions on how to use Threshold Clustering. So, potential reasons are dicussed over the next paragraphs.

One of the main differences between ISBSGAll and ISBSG2000 is that ISBSGAll's CC projects are much fewer and older. They amount to 94 projects, covering the period from 1993 to February 1996. ISBSG2000's CC projects form a total of 168 CC projects implemented until the year 2000. It would be reasonable to expect that the fewer and older CC projects could lead to CC models whose suitability varies more over time. In such cases, a more frequent collection of WC training projects can potentially help to better track the best Dycom instances.

However, the experiments also suggest that the base learner being used can avoid fewer and older CC projects to cause such larger variability in the predictive performance. In particular, LogLR was a better base learner for Dycom when the CC projects were fewer and older. It led to less performance variability and did not require the collection of a larger number of WC training projects. This means that, depending on the base learner and its error variability, it is not necessary to collect more and more recent WC training projects. For the datasets with more and more recent CC projects, RT and BagRT obtained better results as base learners of Threshold Clustering, as indicated in Tables 1, 2 and 3. A valuable future work would be to investigate whether such findings are generalisable to other datasets, when they become available.

RQ4: Threshold Clustering was less successful in reducing the number of required WC training projects when the predictive performance of the Dycom instances used by the online supervised tuning procedure varied more. This occurred when using RT or BagRT in the scenario with fewer and older CC projects. Increasing the frequency with which WC training projects are collected can potentially help the tuning procedure to better track the best Dycom instance to use in such case. However, the use of LogLR as base learner was able to avoid collecting more WC training projects, by leading to more stable predictive performance in this scenario. It is thus recommended for it. For scenarios with newer and more abundant CC projects, RT and BagRT were better base learners for Threshold Clustering.

13 Further Considerations

This section presents further considerations in terms of running time (Section 13.1) and overfitting (Section 13.2).

13.1 Running Time

Table 9 shows the running times in seconds for WC, Threshold Clustering without tuning the number of CC splits, and Threshold Clustering with the proposed online supervised parameter tuning procedure. The algorithms were run on a 2.9 GHz Intel Core i5 with 16 GB and macOS High Sierra. The running time reported for Threshold Clustering without the proposed online supervised parameter tuning procedure is the average running time with the $\lfloor N/10 \rfloor$ different cluster numbers investigated in the study, where N is the size of the CC dataset. The running times reported for BagRT are based on the average across 30 runs. All running times consider the time taken for training plus the time taken for predicting the whole dataset.

Table 9 Running time in seconds

Base learner	Method	ISBSG2000	ISBSG2001	ISBSGAll
RT	WC	0.25	0.12	0.39
	Threshold Clustering without tuning	0.10	0.10	0.20
	Threshold Clustering with tuning	1.55	2.09	1.78
BagRT	WC	2.39	1.18	4.61
	Threshold Clustering without tuning	0.61	0.54	0.54
	Threshold Clustering with tuning	9.82	11.80	4.90
LogLR	WC	0.22	0.14	0.36
	Threshold Clustering without tuning	0.13	0.11	0.15
	Threshold Clustering with tuning	2.09	2.5	1.34

We can see that the time taken to run Threshold Clustering without CC split tuning is smaller than the time taken to run WC. This is because, despite Threshold Clustering having learners for each CC subset, it is trained on only a tenth of the WC projects. We can also see that Threshold Clustering with CC split tuning took roughly $\lfloor N/10 \rfloor$ times longer than Threshold Clustering without CC split tuning to run. This is in line with the fact that the proposed online supervised parameter tuning procedure runs $\lfloor N/10 \rfloor$ instances of Threshold Clustering. The running time of Threshold Clustering with CC split tuning was also larger than that of WC.

The time taken to run Threshold Clustering with CC split tuning could be reduced by running each instance of Threshold Clustering in parallel. However, it is reasonable to say that this would not be necessary – the runtime of all approaches can be considered negligible as the rate of incoming projects is far much smaller than the maximum running time of 11.80 second shown in Table 9. It is also worth noting that the heaviest computational cost comes from training. The time taken for making N predictions at the end of the whole training process was always smaller than 0.1 s for all approaches, meaning that any of these approaches can provide very fast predictions whenever a new project has to be estimated.

13.2 Overfitting

When the proposed online hyperparameter tuning procedure uses a given WC training project for estimating the predictive performance of a given instance of Dycom, this is done *before* this instance is trained on this training project. Therefore, the estimation of the predictive performance used for tuning works as a validation error, rather than training error. This helps to avoid overfitting (Corazza et al. 2013).

Overfitting is also less likely to occur in non-stationary environments, where the data generating process suffers changes over time. This is because the examples used for parameter tuning are likely to come from different distributions over time. As we learn over time, when there is overfitting, one would expect to see an initial decrease in the test error, followed by an increase as more training data is used. This does not happen in Fig. 6. Instead, we see several spikes of test error followed by decreases in test error, in line with what one would expect when there are changes in the underlying data generating process (Gama et al. 2004).

The fact that the proposed online supervised tuning procedure does not use the training error for tuning, the trend in the errors obtained by Dycom with the proposed online hyperparameter tuning procedure over time, and the successful Threshold Clustering results in terms of improving predictive performance with respect to WC approaches trained on the same number of WC projects give us confidence that there are no detrimental levels of overfitting.

14 Threats to Validity

When using machine learning approaches, it is important that the approaches being compared use fair hyperparameter choices to address internal validity (Menzies and Shepperd 2012; Song et al. 2013). In this paper, the base learners used as WC models and within Dycom were either hyperparameterless, or were shown not to be much affected by hyperparameter choice in previous work (Song et al. 2013). This avoids unfair comparisons. Dycom has two extra hyperparameters (β and lr) which were set to the same values for all datasets used in this study, i.e., they were not fine tuned for each dataset and therefore should not lead to an unfair advantage to Dycom over the WC models. This paper also investigates a procedure that eliminates one hyperparameter from Dycom's design—the number of CC subsets. This hyperparameter is automatically chosen by the proposed tuning procedure, which turns this hyperparameter into an internal parameter automatically learnt by the overall Dycom approach.

The choice of base learner to be used with Dycom was shown to affect its predictive performance in this work. One could possibly consider the type of base learner as a hyperparameter. However, a different tuning procedure from the online supervised tuning procedure proposed in this paper would be required to automatically choose the base learner. This is because the same variability that caused the proposed tuning procedure to struggle when using RT and BagRT for ISBSGAll would be present if different base learners had been used to create Dycom instances, making the tuning procedure less effective. Other hyperparameter tuning procedures should be proposed as future work to automatically choose the type of base learner. For now, this study provides insights into the scenarios where each type of base learner could be potentially more adequate (Section 10).

ISBSG2000, ISBSG2001 and ISBSGAll have an overlap of projects. Therefore, these datasets are not independent. However, in online learning, the conclusions that may be obtained with different numbers of preceding and following projects can be very different (Minku and Yao 2017). For example, as shown in Section 9, Dycom with automated tuning managed to achieve similar predictive performance to a WC model for ISBSG2000, but sometimes failed to achieve that for ISBSGAll. Therefore, it is important to include all versions of the ISBSG dataset in the analysis.

In order to address construct validity, this study used AE and LogAE. These are measures unbiased towards over or underestimations, being recommended for SEE studies (Shepperd and McDonell 2012). LogAE is also independent of project size. SA (Shepperd and McDonell 2012) was used to give a better idea of the magnitude of the differences in predictive performance and of the impact that they are likely to have in practice. Scott-Knott procedure with bootstrap test was used to check the statistical significance of the differences in predictive performance. Effect size A12 was used to support the comparisons done by the Scott-Knott procedure. Moreover, the predictive performance was time-decayed, so that it tracks the changes in predictive performance over time, as recommended for studies involving data streams (Gama et al. 2009).

In terms of external validity, besides never using a WC project for training before using it for testing, three datasets derived from the ISBSG Repository (ISBSG 2011) were investigated in this study. As with other studies involving data, we cannot claim that the results obtained for ISBSG generalise to other datasets. Obtaining additional datasets for this study is difficult due to the need for non-proprietary datasets with information about chronology and about which projects belong to a single-company among the projects of a cross-company dataset. However, the datasets used in this study can be made available through ISBSG (ISBSG 2011). So, researchers and companies willing to use Dycom could use the same CC datasets used in this study.

Even though the most successful approach in drastically reducing the number of WC projects identified in RQ1 is not guaranteed to be the same for other datasets, the answer to RQ2 provides an explanation of the characteristics of the CC datasets that made Threshold Clustering more suitable than other clustering methods, contributing to external validity. In particular, Threshold Clustering is likely to remain suitable for other CC datasets that are also exponentially distributed and not clustered. RQ3 shows that the proposed online supervised hyperparameter tuning approach is not always successful, and RQ4 complements it by explaining why that was the case, again contributing to external validity. In particular, it is likely that the proposed online supervised hyperparameter tuning procedure will work well in cases where the predictive performance of the Dycom instances does not vary wildly over time, whereas this approach is likely to fail in cases where there are wild variations.

It is worth noting that this paper investigated cross-company SEE in an online learning scenario, where the data generating process may change over time. Changes in the ISBSG data used in this study are very likely to have ocurred between 1993 and 2003 (Minku and Yao 2017), meaning that the approaches have been evaluated under changing conditions in this paper. As it is likely that changes would also happen in future data, approaches designed to cope with changes such as the one proposed in this paper are likely to remain useful over time.

Dycom considers CC projects as fixed CC datasets available prior to learning, as in previous work Minku and Yao (2017, 2014). Such fixed CC datasets can be useful for prolonged periods of time, as shown in Minku and Yao (2017). Therefore, Dycom as investigated here is applicable in practice. If the weights of all CC mapped models become too low for a prolonged period of time, this may be an indication that Dycom needs to be re-built with updated CC datasets. However, the results obtained in this and previous work involving Dycom (Minku and Yao 2014) indicate that Dycom does not need to be reset often.

As with other machine learning studies involving ensembles, we cannot claim that the results obtained with RTs, BagRTs, and LogLR generalise to other base learners. However, this threat is mitigated by examining the conditions under which the approach is or is not recommended to be used (Section 12). In particular, if the errors of the adopted base learners do not wildly vary over time, it is likely that the proposed online supervised hyperparameter tuning approach will work well.

15 Conclusions

Dycom is a promising online supervised learning approach for SEE. It is able to reduce the amount of WC training projects required for training SEE models while maintaining or improving the predictive performance of a corresponding WC approach. However, its good predictive performance depends on the choice of the number of CC subsets (and the CC subsets themselves) to create its CC models. A poor choice can lead to worse predictive performance than WC approaches. This paper therefore proposed the first procedure for automatically tuning hyperparameters in online supervised learning, and investigated whether clustering methods can help to create good CC splits for use with Dycom when used in combination with automated procedures to choose the number of CC subsets.

Six different clustering methods were investigated, along with three different matching tuning procedures for the number of CC subsets. The experiments show that a very simple clustering procedure (Threshold Clustering) associated with the proposed online supervised tuning procedure was the most successful in improving MdAE and MdLogAE when a limited number of WC training projects was available (RQ1). In particular, it managed to produce up to 36% better MdAE and up to 30% better MdLogAE than the median baseline (Median-P10). It also obtained SA_{AE} up to 40% better and $SA_{Log_{AE}}$ up to 25% better than WC models trained with a limited number of WC projects (WC-P10). Threshold Clustering was more adequate than other clustering methods because the CC training projects were exponentially distributed and not really clustered (RQ2).

Most of the time (in 15 out of 18 cases), Threshold Clustering with the proposed online supervised tuning procedure was successful in enabling Dycom to use 10 times less WC projects than a corresponding WC model while maintaining or even improving MdAE and

MdLogAE (RQ3). The 3 out of 18 cases in which this approach failed in achieving that had fewer and older CC projects, which led to higher variability in the predictive performance of the Dycom instances being monitored by the online supervised tuning procedure when using RTs or BagRTs as the base learners. Increasing the frequency with which WC training projects are collected can potentially help the tuning procedure to better track the best Dycom instance to use in such scenario. However, the use of LogLR led to more stable predictive performance in this scenario, avoiding the need to increase the number of WC training projects (RQ4).

Overall, the proposed online supervised tuning procedure was generally successful in enabling a very simple threshold-based clustering approach to obtain the most competitive results. This demonstrates the value of automatically tuning hyperparameters over time in a supervised way. Future work includes an investigation of procedures for automatically deciding the number of WC training projects to be collected for training, as well as for choosing the best base learner to use. An extension of this study with more datasets is a valuable future direction when suitable datasets become available, and so is a case study with industry. mAn investigation of Dycom and the proposed online hyperparameter tuning approach in the context of Agile estimation would also form a valuable future contribution.

Acknowledgements The author is thankful to Siqing Hou for the implementation of the initial version of the code to integrate Dycom with WEKA's clustering algorithms, which was created during his internship at the University of Birmingham (UK). The author is thankful to Tim Menzies, for sharing his Scott-Knott and Bootstrap implementation, as well as his latex commands for creating quartile plots. The author is thankful to Luigi Dragone, for making his Spectral Clustering code available. The author is also thankful to the editors and anonymous reviewers, for their constructive comments. This work was supported by EPSRC Grant No. EP/R006660/1.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

References

Agrawal A, Menzies T (2018) Is "better data" better than "better data miners"? In: International conference on software engineering (ICSE), pp 1050–1061

Amasaki S, Takahara Y, Yokogawa T (2011) Performance evaluation of windowing approach on effort estimation by analogy. In: IWSM-MENSURA, pp 188–195

Bishop C (2006) Pattern recognition and machine learning. Springer, Berlin

Boehm B (1981) Software engineering economics. Prentice-Hall, Englewood Cliffs

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Briand L, Langley T, Wieczorek I (2000) A replicated assessment of common software cost estimation techniques. In: International conference on software engineering (ICSE). Como, Italy, pp 377–386

Cohen J (1992) A power primer. Psychol Bull 112:155–159

Corazza A, Di Martino S, Ferrucci F, Gravino C, Sarro F, Mendes E (2013) Using tabu search to configure support vector regression for effort estimation. Empir Softw Eng J (EMSE) 18(3):506–546

Dejaeger K, Verbeke W, Martens D, Baesens B (2012) Data mining techniques for software effort estimation: a comparative study. IEEE Trans Softw Eng (TSE) 38(2):375–397

Ditzler G, Roveri M, Alippi C, Polikar R (2015) Learning in nonstationary environments: a survey. IEEE Comput Intell Mag (CIM) 10(4):12–25

Fu W, Menzies T (2017) Easy over hard: a case study on deep learning. In: Symposium on the foundations of software engineering (FSE), pp 49–60

Fu W, Menzies T, Shen X (2016) Tuning for software analytics: is it really necessary? Inf Softw Technol (IST) 76:135–146

Gallego J, Rodriguez D, Sicilia M, Rubio M, Crespo A (2007) Software project effort estimation based on multiple parametric models generated through data clustering. J Comput Sci Technol (JCST) 22(3):371–378

Gama J, Medas P, Castillo G, Rodrigues P (2004) Learning with drift detection. In: Proceedings of the 7th Brazilian symposium on artificial intelligence (SBIA'04) - Lecture notes in computer science, vol 3171. Springer, São Luiz do Maranhão, pp 286–295

Gama J, Sebastiao R, Rodrigues P (2009) Issues in evaluation of stream learning algorithms. In: ACM SIGKDD conference on knowledge discovery and data mining (KDD), pp 329–337

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. SIGKDD Explor 11(1):10–18

Huang SJ, Chiu NH, Liu YJ (2008) A comparative evaluation on the accuracies of software effort estimates from clustered data. Inf Softw Technol (IST) 50:879–888

ISBSG (2011) The international software benchmarking standards group. http://www.isbsg.org

Jeffery R, Ruhe M, Wieczorek I (2010) A comparative study of two software development cost modeling techniques using multi-organizational and company-specific data. Inf Softw Technol (IST) 42(14):1009–1016

Jørgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Trans Softw Eng (TSE) 33(1):33–53

Kannan R, Vempala S, Vetta A (2000) On clusterings—good, bad and spectral. Tech. rep., Yale University

Kitchenham B, Mendes E (2004) A comparison of cross-company and within-company effort estimation models for web applications. In: METRICS, Chicago, pp 348–357

Kitchenham B, Mendes E, Travassos G (2007) Cross versus within-company cost estimation studies: a systematic review. IEEE Trans Softw Eng (TSE) 33(5):316–329

Kocaguneli E, Cukic B, Menzies T, Lu H (2013) Building a second opinion: learning cross-company data. In: International conference on predictive models in software engineering (PROMISE), pp 12.1–10

Kocaguneli E, Gay G, Menzies T, Yang Y, Keung JW (2010) When to use data from other projects for effort estimation. In: Automated software engineering (ASE). Antwerp, Belgium, pp 321–324

Kocaguneli E, Menzies T, Bener A, Keung JW (2012) Exploiting the essential assumptions of analogy-based effort estimation. IEEE Trans Softw Eng (TSE) 38(2):425–438

Kocaguneli E, Menzies T, Mendes E (2015) Transfer learning in effort estimation. EMSE 20(3):813–843

Lavesson N, Davidsson P (2006) Quantifying the impact of learning algorithm parameter tuning. In: AAAI conference on artificial intelligence

Lefley M, Shepperd M (2003) Using genetic programming to improve software effort estimation based on general data sets. In: Genetic and evolutionary computation conference (GECCO), vol LNCS 2724, Chicago, pp 2477–2487

Lokan C, Mendes E (2017) Investigating the use of moving windows to improve software effort prediction: a replicated study. Empir Softw Eng J (EMSE) 22:716–767

McDonell S, Shepperd M (2007) Comparing local and global software effort estimation models—reflections on a systematic review. In: ESEM, Madrid, pp 401–409

Menzies T, Shepperd M (2012) Special issue on repeatable results in software engineering prediction. EMSE 17:1–17

Menzies T, Butcher A, Cok D, Marcus A, Layman L, Shull F, Turhan B, Zimmerman T (2013) Local vs. global lessons for defect prediction and effort estimation. IEEE Trans Softw Eng (TSE) 39(6):822–834

Menzies T, Krishna R, Pryor D (2017) The seacraft repository of empirical software engineering data. tiny.cc/seacraft

Menzies T, Yang Y, Mathew G, Boehm B, Hihn J (2017) Negative results for software effort estimation. Empir Softw Eng J (EMSE) 22(5):2658–2683

Minku L (2016) On the terms within- and cross-company in software effort estimation. In: International conference on predictive models and data analytics in software engineering (PROMISE). Ciudad Real, Spain, pp 4.1–4.4

Minku L (2018) Oates: a fully dynamic transfer learning approach for software effort estimation (under review)

Minku L, Hou S (2017) Clustering Dycom: an online cross-company software effort estimation study. In: International conference on predictive models and data analytics in software engineering (PROMISE), pp 12–21

Minku L, Yao X (2012) Can cross-company data improve performance in software effort estimation? In: International conference on predictive models in software engineering (PROMISE), Lund, pp 69–78

Minku L, Yao X (2013) Ensembles and locality: insight on improving software effort estimation. Inf Softw Technol (IST) 55(8):1512–1528

Minku L, Yao X (2014) How to make best use of cross-company data in software effort estimation? In: International conference on software engineering (ICSE), Hyderabad, pp 446–456

Minku L, Yao X (2017) Which models of the past are relevant to the present? A software effort estimation approach to exploiting useful past models. Automat Softw Eng J 24(3):499–542

Minku L, Sarro F, Mendes E, Ferrucci F (2015) How to make best use of cross-company data for web effort estimation? In: International symposium on empirical software engineering and measurement (ESEM). Bergamo, Italy

Mittas N, Angelis L (2013) Ranking and clustering software cost estimation models through a multiple comparisons algorithm. IEEE Trans Softw Eng (TSE) 39(4):537–551

Nair V, Agrawal A, Chen J, Fu W, Mathew G, Menzies T, Minku L, Wagner M, Yu Z (2018) Data-driven search-based software engineering. In: Mining software repositories (MSR), pp 341–352

Oliveira A, Braga P, Lima R, amd Cornelio R (2010) GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. Inf Softw Technol (IST) 52:1155–1166

Pelleg D, Moore A (2000) X-means: extending k-means with efficient estimation of the number of clusters. In: International conference on machine learning (ICML), pp 727–734

Rokach L, Maimon O (2005) Clustering methods. Springer, Berlin, pp 321–352

Sarro F, Petrozziello A (2018) Linear programming as a baseline for software effort estimation. In: ACM transactions on software engineering and methodology (TOSEM) (in press)

Sarro F, Petrozziello A, Harman M (2016) Multi-objective software effort estimation. In: International conference on software engineering (ICSE), pp 619–630

Shepperd M, McDonell S (2012) Evaluating prediction systems in software project estimation. Inf Softw Technol (IST) 54(8):820–827

Shepperd M, Schofield C (1997) Estimating software project effort using analogies. IEEE Trans Softw Eng (TSE) 23(12):736–743

Shi J, Malik J (2000) Normalized cuts and image segmentation. IEEE Trans Pattern Anal Mach Intell 22(8):888–905

Song L, Minku L, Yao X (2013) The impact of parameter tuning on software effort estimation using learning machines. In: International conference on predictive models and data analytics in software engineering (PROMISE), Article No. 9, 10p. Baltimore

Song L, Minku L, Yao X (2018) A novel automated approach for software effort estimation based on data augmentation. In: Symposium on the foundations of software engineering (FSE) (accepted)

Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2016) Automated parameter optimization of classification techniques for defect prediction models. In: International conference on software engineering (ICSE)

Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2017) An empirical comparison of model validation techniques for defect prediction models. IEEE Trans Softw Eng (TSE) 43(1):1–18

Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2018) The impact of automated parameter optimization for defect prediction models. IEEE Trans Softw Eng (TSE) (in press)

Turhan B, Mendes E (2014) A comparison of cross- versus single- company effort prediction models for web projects. In: SEAA, Verona, pp 285–292

Usman M, Mendes E, Weidt F, Brito R (2014) Effort estimation in agile software development: a systematic literature review. In: Proceedings of the 10th international conference on predictive models in software engineering, pp 82–91

Vargha A, Delaney HD (2000) A critique and improvement of the cl common language effect size statistics of Mcgraw and Wong. J Educ Behav Stat 25:101–132

Wang S, Minku L, Yao X (2015) Resampling-based ensemble methods for online class imbalance learning. IEEE Trans Knowl Data Eng 27:1356–1368

Wieczorek I, Ruhe M (2002) How valuable is company-specific data compared to multi-company data for software cost estimation? In: METRICS, Ottawa, pp 237–246

Xia T, Chen J, Mathew G, Shen X, Menzies T (2018) Why software effort estimation needs SBSE. arXiv:1804.00626v1

Xia T, Krishna R, Chen J, Mathew G, Shen X, Menzies T (2018) Hyperparameter optimization for effort estimation. arXiv:1805.00336v2

Dr. Leandro L. Minku is a Lecturer in Intelligent Systems at the School of Computer Science, University of Birmingham (UK). Prior to that, he was a Lecturer in Computer Science at the University of Leicester (UK), and a research fellow at the University of Birmingham (UK). He received the PhD degree in Computer Science from the University of Birmingham (UK) in 2010. Dr. Minku's main research interests are machine learning for software engineering, search-based software engineering, machine learning for non-stationary environments / data stream mining, and ensembles of learning machines. His work has been published in internationally renowned journals such as IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology, IEEE Transactions on Knowledge and Data Engineering, and IEEE Transactions on Neural Networks and Learning Systems. Among other roles, Dr. Minku is the general chair for the International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2019 and 2020), the co-chair for the Artifacts Evaluation Track at the International Conference on Software Engineering (ICSE 2020), an associate editor for the Journal of Systems and Software, an editorial board member for Neurocomputing and a conference correspondent for IEEE Software.