Empirical Software Engineering
https://doi.org/10.1007/s10664-018-9647-0

An ensemble-based model for predicting agile software development effort

Onkar Malgonde[1] · **Kaushal Chari**[2]

Published online: 11 September 2018

Abstract
To support agile software development projects, an array of tools and systems is available to plan, design, track, and manage the development process. In this paper, we explore a critical aspect of agile development i.e., effort prediction, that cuts across these tools and agile project teams. Accurate effort prediction can improve the planning of a sprint by enabling optimal assignments of both stories and developers. We develop a model for story-effort prediction using variables that are readily available when a story is created. We use seven predictive algorithms to predict a story's effort. Interestingly, none of the predictive algorithms consistently outperforms others in predicting story effort across our test data of 423 stories. We develop an ensemble-based method based on our model for predicting story effort. We conduct computational experiments to show that our ensemble-based approach performs better in comparison to other ensemble-based benchmarking approaches. We then demonstrate the practical application of our predictive model and our ensemble-based approach by optimizing sprint planning for two projects from our dataset using an optimization model.

Keywords Agile · Effort prediction · Ensemble · Machine learning · Scrum · Sprint planning

1 Introduction

Developing software under budget and within the planned time line in an agile software development environment relies in part on the development team's ability to predict the

Communicated by: Martin Shepperd

✉ Onkar Malgonde
omalgonde@niu.edu

Kaushal Chari
kchari@usf.edu

[1] Operations Management & Information Systems, College of Business, Northern Illinois University, DeKalb, IL, USA

[2] Information Systems and Decision Sciences, Muma College of Business, University of South Florida, Tampa, FL, USA

number of person-hours (effort) that project developers will need to complete a project. This requires accurate estimations of stories that constitute a project. A story could include one or more features which the user has requested (for example, one feature of the agile software might be "update user's address"). These stories are developed in "sprints," or iterations of software development cycles. Accurately estimating the effort (number of person-hours) a story will require is crucial for the following reasons. First, it determines which stories the development team selects at a given sprint to maximize the project's velocity (Hearty et al. 2009). Second, accurate estimates can be used to optimize individual developer effort across multiple projects, that the developer is working on during the sprint period. Third, optimal scheduling of developers and stories for a portfolio of projects could be done to achieve global efficiencies.

In current practice, the development team itself may predict the amount of effort a project's story will require (Usman et al. 2014), but such team estimates are often inaccurate (Hussain et al. 2013). In a survey of human-effort estimation techniques for agile software development, Usman et al. (2015) report that "according to 52% of the respondents, effort estimates are out of range by [a] factor of 25% or more."

Another approach to estimating effort for agile software development is the use of machine learning algorithms that run project-related data (e.g., product backlog, stories, sizes, priorities, subtasks, skills). But these predictive models consider only a limited set of algorithms and features (Usman et al. 2014). Further, none of the current machine learning algorithms consistently outperforms others in predicting effort for agile software development projects (see computational results), thus exhaustive pre-testing is required to select the best model for a particular data set.

To resolve these limitations in current methods for predicting effort, we have developed an ensemble-based approach, incorporating predictive algorithms, using readily available data and requiring no a-priori testing to determine suitability for a particular data set. We have also developed a predictive model and an optimization model for effort prediction. We focus on the Scrum software development framework, already used by many practitioners.

For over a decade, agile software development has had multiple tools to plan, design, track, and manage the development process. For example, the Dynamic Systems Development Method (DSDM) provides a framework for rapid application development (Stapleton 1997); Crystal, a family of methodologies, focuses on people rather than the processes followed in the project (Abrahamsson et al. 2002); Feature-driven Development (FDD) emphasizes a product's features (Palmer and Felsing 2002); Adaptive Software Development (ASD) enables adaptation to the emergent state of the project; Scrum provides a framework to manage product development and project flow (Schwaber and Sutherland 2016); and Extreme Programming (XP) focuses on development activities with short cycles (Beck and Andres 2004). Various approaches are flexible enough to respond to changes in customers' requirements, not just reactively but even proactively (Conboy 2009).

Scrum and XP are the most widely adopted agile approaches (Pikkarainen et al. 2008). Based on a survey of 3,880 respondents, VersionOne (2016) reports that 58% used Scrum. Diverse industries have adopted Scrum for projects with varying sizes and goals. These dynamics make it a good basis for our method. Scrum is an iterative and incremental methodology for agile software development, characterized by sprints that are typically time boxed to 1-3 weeks each. Each sprint constitutes one development cycle that incorporates the planning, development, testing, integration, and creation of a working deliverable. Overarching the entire project is the product backlog. The product backlog keeps track of all user stories which represent user requirements. In the Scrum methodology, the development team is required to plan each sprint. Planning involves calibrating sprint estimates,

selecting appropriate stories, assigning developers, estimating effort required for stories by incorporating historic performance of the team (prior sprints, if any).

In this paper, we propose a predictive model and an ensemble-based approach to predict the effort required to develop a story. Specifically, we use readily available data during the planning phase of a sprint to predict the required development effort for each story in person-hours. Predictive models in agile software development environments should meet the following points: (a) a model's predictive accuracy should be better than the development team's estimates, (b) a model should have the ability to utilize readily available data, (c) a model should be easy to deploy for sprint planning, and (d) a model should be maintainable.[1] To meet these points, we employ machine learning algorithms that perform better than team estimates for effort prediction, as computational results later in the paper will show. Further, as none of the machine learning algorithms consistently outperforms others in predicting effort for agile software development projects (see computational results), individual algorithms require exhaustive pre-testing to select the best model for a particular data set. We therefore propose an ensemble-based approach that could be used with confidence to provide good predictions without the need for a-priori testing. Our ensemble-based approach, on average, outperforms team estimation and other ensemble-based approaches. An ensemble consists of n predictive algorithms that are trained using the same data set. Based on the training dataset, each predictive algorithm trains its model. The trained model is then used to predict future stories. The ensemble aggregates each item's prediction using weights assigned to each item. With every prediction, the ensemble can learn an item's predictive accuracy and can adjust item weights accordingly. The ensemble's prediction for each story is then provided to a novel integer programming model, which optimizes available effort across the team to generate a schedule which maximizes the story points developed.

This paper contributes to the agile software effort prediction literature by proposing three components: (a) a predictive model to predict story effort, (b) an ensemble-based approach to aggregate predictions, and (c) an optimization model to optimize available effort. In what immediately follows, we discuss related work in agile software development effort estimation and ensemble-based approaches. In Section 3, we describe the real-world dataset used in our experiments. In Section 4, we discuss our predictive model. In Section 5, we present the machine learning algorithms and experiments performed using the real data. In Section 6, we present an ensemble-based approach for prediction, and then computational results in Section 7. In Section 8, we demonstrate the practical application of our proposed ensemble-based approach in sprint planning in the context of two real-world projects. We then discuss the threats to validity in Section 9, followed by the conclusion, limitations, and future research directions in Section 10.

2 Related Work

2.1 Effort Estimation in Agile Software Development

The general nature of effort estimation has been studied in software engineering (Dejaeger et al. 2012; Jørgensen and Shepperd 2007), but agile software development's iterative nature requires specialized techniques. Traditionally, these techniques (a) have been based on

[1] We thank the anonymous reviewer for pointing this out.

human judgement, and (b) use data to estimate the effort required. Table 1 provides a brief summary of these studies. For human judgement, *Expert judgement* is one of most popular techniques in software effort estimation (Jørgensen and Shepperd 2007; Usman et al. 2014),

Table 1 Related work on agile effort estimation

Study	Dependent Variable	Independent Variables	Prediction approach	Data (projects)	Findings
Abrahamsson et al. (2007)	Total Coding effort per class	Weighted methods per class, coupling between object classes	Regression, Neural Networks	2	Incremental models outperform global models in effort prediction.
Abrahamsson et al. (2011)	Effort (in hours)	Character count, keywords, priority	Expert judgement, Regression, Neural Networks, Support Vector Machine	1337 stories (2 projects)	Effort prediction improves when stories are structured.
Hussain et al. (2013)	Effort class	Keywords	Text Mining	4	Subjective judgement of size is often erroneous.
Logue et al. (2007)	Size Estimate in days	Expert review of story	Expert judgement	6 stories (28 subtasks)	Expert judgement can be used for sprint planning.
Haugen (2006)	Effort per story	Story cards from predicting group	Expert judgement, Planning Poker	50 stories	Planning Poker has lower prediction error than unstructured group estimation.
Mahnic and Hovelja (2012)	Effort per story	Story description	Planning Poker	60 stories	Planning Poker introduced overoptimism in estimates.
Santana et al. (2011)	Correlation between story points and function points	Story points and function points	Planning Poker	2191 stories	Function points can provide story points.
Nunes et al. (2011)	Size estimation	Use Cases	Use Case Points (UCP) method	14 student groups	Use Case Points method's size estimations are consistent for use case complexity.
Current paper	Effort per story (in hours)	Priority, number of subtasks, size, developer experience, sprint	Bayesian Network, Ridge Regression, Linear Regression, Neural Network, Support Vector Machine, Decision Trees, K-Nearest Neighbors, AdaBoost, Random Forest, Average, Gradient Boosting, Extra Trees, Stacking	503 stories (24 projects)	Ensemble-based approach could provide superior effort predictions. Sprint planning can be optimized with machine learning and optimization techniques.

the effort is estimated by an individual with knowledge and experience in the project and related business domain. Another estimation technique is *Planning Poker* (Grenning 2002; Santana et al. 2011). In this gamified estimation technique, team members individually assess the effort required to develop a particular story and raise a card denoting the estimate (Haugen 2006; Mahnic and Hovelja 2012). This allows each team member to participate in the estimation process, thereby discouraging bias in the estimation. Finally, the Use Case Points (UCP) (Karner 1993) approach uses modeling diagrams to estimate the complexities in actors, use cases, and the environment to estimate the effort required. Extending the UCP method, Nunes et al. (2011) introduce requirements complexity to estimate the effort required.

Human approaches for effort estimation may introduce bias and intentional distortions. Magazinius et al. (2012) report that stakeholders may intentionally distort estimates depending on the contextual factors. Distortions may be introduced to increase available time, gold-plate features, hide incompetence, or accommodate emergent requests. This leads to downstream consequences on project's budget, time line, and quality.

Empirical studies in agile effort estimation have employed machine learning algorithms for effort estimation. Abrahamsson et al. (2007) develop linear regression and neural network models from object-oriented design metrics. These models show improvements in the prediction accuracy of models that consider iterative development. Abrahamsson et al. (2011) extract keywords from story descriptions to develop a linear model, which is benchmarked using team estimations. Similarly, Hussain et al. (2013) extract keywords from informally written story descriptions to estimate story size. Across these studies, effort prediction has typically been measured in person-hours required to develop a story. Furthermore, story descriptions have been text-mined to extract important keywords that can improve the predictive accuracy of the models. Regression and neural network approaches have been commonly used for effort prediction, along with techniques like support vector machines and text mining. For a systematic review of effort estimation in agile software projects, see Usman et al. (2014, 2015).

Prior work in software effort estimation, but not in the agile context, has considered the chronology of projects that can be used as training data. Across four studies, Lokan and Mendes (2014) find that limiting training data to specific recent projects or time window does not provide superior estimation accuracy for a new project. However, MacDonell and Shepperd (2010) conclude that failure to account for the temporal nature of software development projects may lead to unreliable estimates.

Prior studies in agile software effort estimation that use machine learning approaches have three limitations that we address in this paper. First, these studies do not incorporate temporal ordering of stories across sprints, i.e., they do not require that only stories from sprints 1 and 2 be used in the training data to predict a story from sprint 3. A fundamental characteristic of the agile approach is learning (Conboy 2009), which could manifest itself in terms of increased efficiency across sprints. Second, machine learning algorithms have not been fully explored in estimating effort in agile software development projects. Third, prior studies have not considered some of the readily available project data such as those used in the current study in estimating agile project effort.

2.2 Ensembles of Predictive Algorithms

Given a feature vector χ for a story, the goal is to predict the effort in person-hours. Individual predictive algorithms (which have been trained using training data) provide predictions for the new feature vector χ. The ensemble then incorporates the predictive algorithms'

predictions to determine the final prediction based on weights which determine the influence of each predictive algorithm. Algorithms with higher weights have a greater influence on the ensemble prediction.

Literature in software engineering that considers ensemble-based approaches to aggregate individual estimation techniques and extensive experiments has confirmed the predictive efficacy of ensembles (Kocaguneli et al. 2012). Minku and Yao (2013) compare individual approaches and ensembles across datasets and recommend *bagging* ensembles of regression trees. Another popular ensemble-based approach in the machine learning literature is *boosting*, where weak learners are aggregated to provide final predictions. Azzeh et al. (2015) develop an ensemble-based approach to aggregate different analogy-based adjustment methods and show that their ensemble-based approach provides accurate estimations in comparison to individual adjustment methods. Kultur et al. (2008) develop an ensemble of neural networks with associative memory (ENNA) and show its predictive superiority in comparison to neural networks and regression trees. Similar results are presented by Azhar et al. (2013) in the web effort estimation context. The approaches by Kultur et al. (2008) and Azhar et al. (2013) have limited applicability to effort estimation in the agile software development context for the following reasons: (a) their ensembles consider similar individual contributors, (b) individual algorithms are weak learners, and (c) their ensembles use averaging procedures. Jonsson et al. (2016) use *stacking* (Wolpert 1992) in an automated bug assignment context, where multiple predictive algorithms are combined using a stacked model. Stacking has limited applicability in agile effort estimation as it relies on the efficacy of the stacked model to identify appropriate base learners. Market-based ensemble approaches such as in Perols et al. (2009) address the issues of averaging procedures and the efficacy of stacked models by giving more importance to strong individual contributors by adjusting their weights based on performances.

The classical approaches for ensemble predictions use average, weighted average, and majority voting. In Perols et al. (2009), the authors propose a market-based ensemble that incorporates multiple classifier decisions to provide superior classifications for a two-class decision problem. However, in the current paper, the dependent variable is continuous. Jahedpari (2016) develop an artificial market-based prediction mechanism for continuous variables where the primary focus is on aggregating predictions from disparate data sources. Each predictive algorithm in the ensemble, i.e., market participant, using its dataset, provides a prediction and associated investments, which serve as weights. A reward function is then used to determine participants' payoffs. This approach has the following characteristics: (a) multiple parameters (up to four) need to be calibrated, and (b) multiple rounds of bidding are required to establish convergence, before a bid is made, thereby making this approach time consuming. In addition, successive worse performance could deprive a market participant of any capital, which is not desirable as it would eliminate the participant from contributing to predictions in subsequent rounds. Perols et al. (2009) provide capital to all the participants in every round to remedy this problem while using a market-approach for classification. Our ensemble approach uses a single prediction round to determine ensemble prediction. Further, our approach uses fewer parameters (just one parameter is calibrated) and allows individual predictors to increase their influence based on increasingly better performance even when their past performance was poor.

Summarizing the related work, while prior studies in traditional approaches have realized the importance of temporal aspects in software development projects, the literature on agile approaches has considered neither temporal sequencing at the sprint level nor the ensembles of individual predictive algorithms common to traditional approaches. In this study, we consider individual predictive algorithms, ensembles, and temporal sequencing in the

context of agile software development projects. Our proposed ensemble-based algorithm is benchmarked with other ensemble-based approaches from machine learning and software engineering literature. Finally, the predictions are employed in a novel approach to optimize available effort while maximizing story points development.

3 Data

The dataset for this paper was obtained from the information technology (IT) department of a large university. The IT department provides technological support to the university's mission. The data consists of 503 stories from 24 agile software development projects from 2012 to 2016. Application domains for these projects pertained to healthcare and higher education. Development teams were formed from a pool of 14 developers (software professionals paid to develop quality software) for the 503 stories. Some projects had the same development team. Team members were employees of the university and were required to follow industry standards and processes. Each software development team member was required to report effort expended on project activities. To avoid individual biases, project leads and managers created tasks for stories in the system. As team members completed these tasks, they logged the number of hours expended on the task each day, until the tasks were completed. Table 2 provides the descriptive statistics for the variables.

We categorized variables (Table 3) in our data because (a) some variables were discrete, (b) specific algorithms like the Bayesian network could then be applied (see Table 1), and (c) the predictive accuracy of various individual algorithms could be improved as discretization lowered variances and biases in the data. Specifically, for a given sprint, the number of data points was often low. This often led to poorly trained models with high prediction errors. However, when consecutive sprints were grouped into a category bucket, models could be trained using all the data points within the bucket, thereby leading to superior predictions. We categorized the variables using the equi-width and equi-depth approach (Aggarwal 2015). Category widths for effort, sprint, and subtasks (except for when a story had zero subtasks) were determined to ensure even distributions of data points across categories. Using a smaller number of categories (larger category size) increased the variance in each category which resulted in worse prediction performance, as different stories were grouped in a category bucket.

Table 2 Descriptive statistics for the data (24 projects; 503 stories)

Variable (unit)	Mean	Median	Std Dev	Min	Max
Effort (in person-hours)	13.81	7.5	15.69	0.15	73.75
Priority*	–	–	–	–	–
Subtasks (number of subtasks)	5.23	2	7.43	0	45
Size (story points)	4.77	5	2.36	1	13
Sprint (sprint number)	9.81	8	7.28	1	34
Programmer 1 Experience (years)	7.32	6	5.33	1	21
Programmer 2 Experience (years)	9.11	7	6.04	1	20
Programmer 3 Experience (years)	7.81	8	1.47	4	10

*Possible values are: Trivial, Minor, Major, Critical

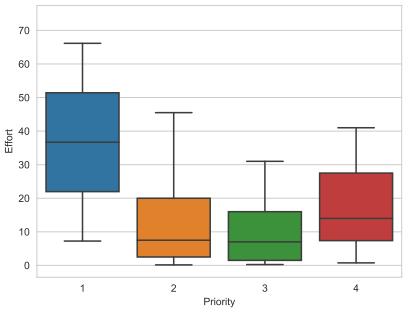
All the algorithms provided point estimates rather than category predictions. Figure 1 provides the distribution of data with boxplots for each predictor variable. For each boxplot, we show the minimum, maximum, first and third quartile, and median effort for each predictor variable's category. Figure 2 illustrates the number of stories in each category across the 503 stories in the dataset. Table 4 provides correlations (Kendall's τ_B) between categorical variables. Multicollinearity using a variance inflation factor (VIf) did not identify any variable with a VIf value greater than 10. We did not

slice the dataset across different projects because (a) projects were homogeneous (supporting the university's mission), (b) similar development standards and processes were followed, and (c) small group of developers were involved. Figure 3 illustrates the heatmap of our dataset where a cell represents the number of stories in a predictor variable's category and effort's category (Fig. 4).

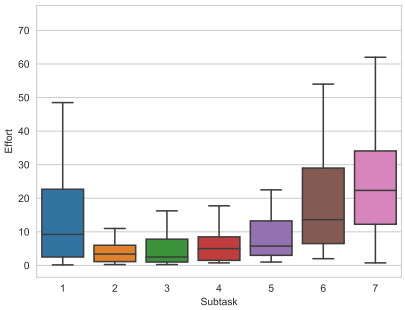
To show the differential impacts of the predictor variables (listed in Table 3) in predicting effort (see Fig. 5), we developed a decision tree to predict the effort required (see Fig. 4). The variables' significance (feature importance, provided by Gini importance of the variable) is noted in Table 5. The interpretation of the decision tree is as follows. For a story whose effort is to be estimated, if the second programmer's experience is less than or equal to 1.5 years, traverse the left node, else the right node. Assuming it is less than 1.5 years, if the size of the story is less than or equal to 3.5 story points, then traverse left. Traversing this tree to the leaf node provides the effort estimate (given by "value" in the leaf node). The primary split in the decision tree is based on the presence of second programmer while the second

Table 3 Predictor variables and their categories

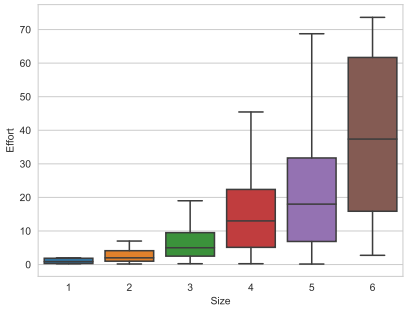
Variable	Categories	Description
Effort (10 categories)	[0,0.99], [1,1.99], [2,3.49], [3.5,5.49], [5.5,7.99], [8,11.99], [12,16.99], [17,24.99], [25,39.99], [40,73.75]	Effort is the number of person-hours required to complete (develop and test) a story
Priority (4 categories)	Trivial (1), Minor (2), Major (3), Critical (4)	Priority of a story, i.e., the relative importance that the customer assigns to each story
Subtasks (7 categories)	0, [1,3], [4,5], [6,7], [8,10], [11,15], [16,45]	Each subtask relates to a specific component (user interface, database, business logic, or other) of the overall story
Size (6 categories)	1, 2, 3, 5, 8, 13	Size is the team's estimate of the relative size of a story with respect to other stories
Sprint (6 categories)	[1,3], [4,5], [6,7], [8,11], [12,15], [16,34]	Sprint in which a story is developed
Programmer 1 Experience (3 categories)	[1,5], [6,10], [15,21]	Number of years of programming experience
Programmer 2 Experience (3 categories)	[1,5], [6,10], [15,21]	Number of years of programming experience
Programmer 3 Experience (3 categories)	[1,5], [6,10], [15,21]	Number of years of programming experience



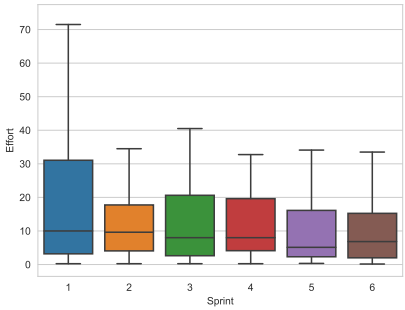
(a) Priority



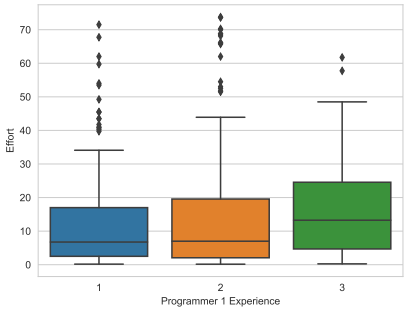
(b) Subtasks



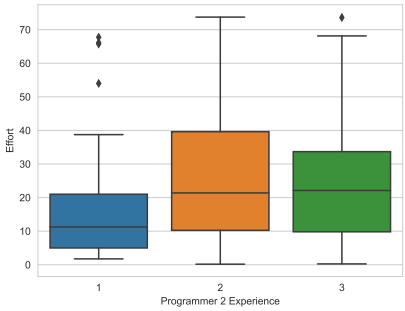
(c) Size



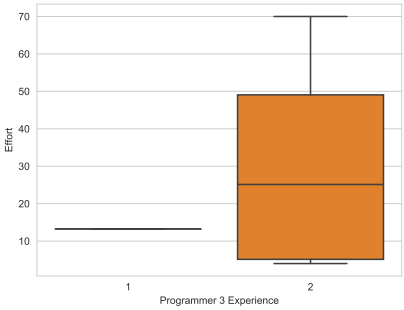
(d) Sprint



(e) Programmer 1 Experience

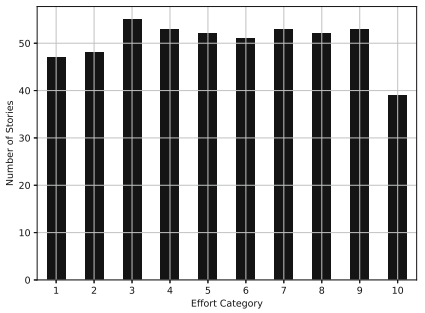


(f) Programmer 2 Experience

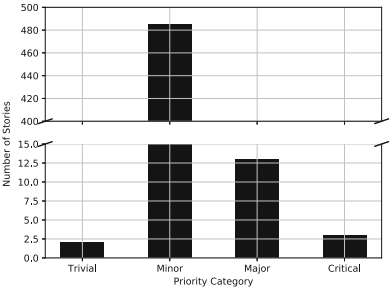


(g) Programmer 3 Experience

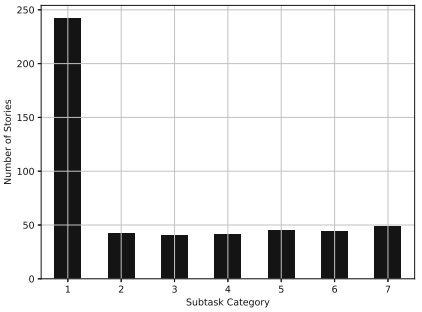
Fig. 1 Boxplot of predictor variables (24 projects; 503 stories)



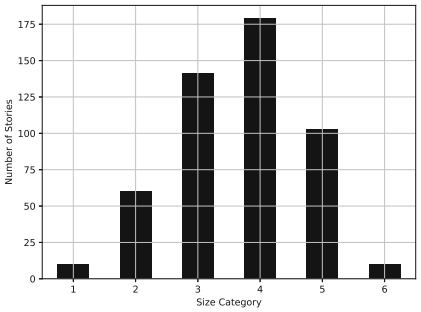
(a) Number of Stories per Effort Category



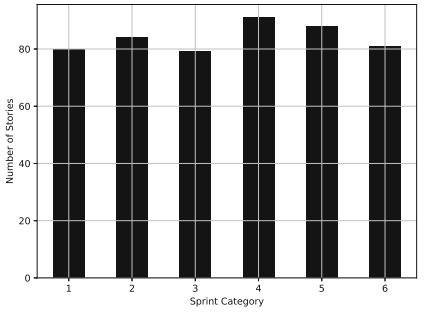
(b) Number of Stories per Priority Category



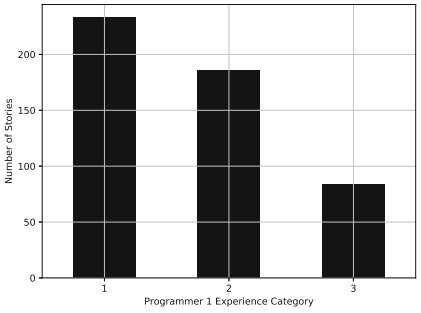
(c) Number of Stories per Subtasks Category



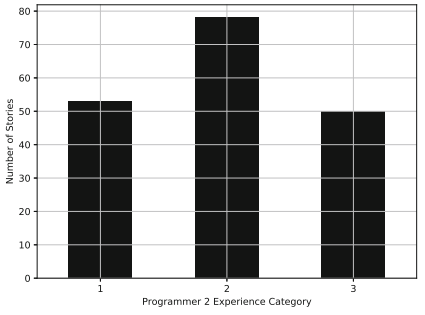
(d) Number of Stories per Size Category

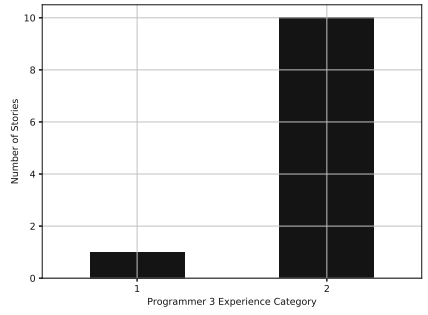


(e) Number of Stories per Sprint Category



(f) Number of Stories per Programmer 1 Experience Category





(g) Number of Stories per Programmer 2 Experience Category (h) Number of Stories per Programmer 3 Experience Category

Fig. 2 Distribution of Variable Categories (24 projects; 503 stories)

Table 4 Correlation (Kendall's τ_B) between predictor variables

		1	2	3	4	5	6	7
1	Effort	1						
2	Priority	–0.03	1					
3	Subtasks	0.09*	–0.02	1				
4	Size	0.46***	0.07	0.1*	1			
5	Sprint	–0.12**	–0.004	0.13*	0.008	1		
6	Programmer 1 Experience	0.09	0.01	–0.17	–0.01	–0.13**	1	
7	Programmer 2 Experience	0.46***	–0.04	–0.08	0.21***	–0.11**	0.03	1
8	Programmer 3 Experience	0.09*	0.05	0.08	0.03	–0.007	0.02	0.19***

*Sig. at 0.05; **Sig. at 0.01; ***Sig. at 0.001

split is based on the size of the story. These variables have greater impact on the predicted effort as seen from Table 5.

4 Predictive Model

Our predictive model (Fig. 5) consists of five independent variables that are readily available when a sprint is planned. The dependent variable is the total development effort (in person-hours) required to develop a story. The five independent variables are (a) *priority* of the story, (b) number of *subtasks* for the story, (c) *size* of the story, (d) *sprint* of the story, and (e) *programmer experience*. Table 3 provides possible values and categories for these variables.

Priority of a story is the relative importance that the customer assigns to each story. The development team prioritizes the development of high priority stories. Each story may include zero or more *subtasks*. Each subtask relates to a specific component (user interface, database, business logic, or other) of the overall story. Creating subtasks provides an efficient way for development teams to maintain cognitive control over larger stories. *Size* of the story is the team's estimate of its size relative to other stories. Stories are typically measured in story points, using the Fibonacci series. *Sprint* of a story is the sprint in which the development of the story is planned and is an integer value. Finally, *programmer experience* is the total programming experience (in years) of a developer assigned to a story. In our dataset, the programming experience, which is self-reported by each developer, considers total programming experience over the programmer's lifetime, and does not differentiate between the technology and the domain of the prior experience. Considering the total programming experience is important, as programmers learn tangible and intangible skills across different projects. For multiple developer instances, our model introduces variables for each additional developer. In summary, our model incorporates human judgement (size estimation of the story) and other story-specific information, as well as developer-specific information available during the sprint planning.

Agile project management tools may log other variables that can be considered in the predictive model, such as performance metrics related to the scrum master and project owner. We do not consider these variables, as either we did not have access to them or they were not consistently available across projects. Papatheocharous et al. (2010) employ a variety of Feature Subset Selection (FSS) algorithms to identify relevant features in cross-company

Fig. 3 Heatmap of predictor variables across effort categories (24 projects; 503 stories)

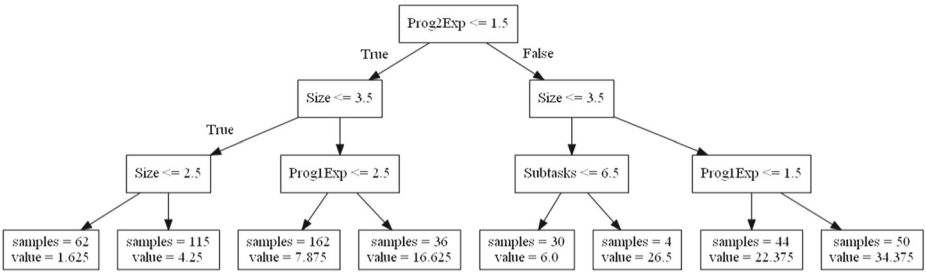


Fig. 4 Decision tree

and within-company datasets in a traditional software development setting. In our agile software development setting, we did not perform feature subset selection at the data collection level because key variables were selected in the model. Feature selection is performed in our experiments to identify important features for each individual algorithm.

5 Experiments

Following the temporal sequencing of sprints in an agile software development project, we used story data from sprint category *n* to predict the effort required to develop stories in sprint category *n+1*, where *n precedes n+1* in the temporal sequencing. After predicting effort for stories in sprint category *n+1*, we used data from sprint categories *n* and *n+1* as training data to predict effort required to develop stories in sprint category *n+2*, and so on, as data from prior iterations are indicators of subsequent iterations (Hearty et al. 2009). Thus, our experiments simulated real-world sprint planning. It should be noted that stories in a sprint category could belong to multiple projects. Our training data consisted of stories from multiple projects.

Cross validation and its variants have been popular techniques to train predictive models in machine learning literature (Shmueli et al. 2016) and effort estimation (Dejaeger et al. 2012). However, a key feature of agile software development projects is that the projects are completed in sequences of sprints. This requires us to follow temporal precedence in training our predictive models. Cross validation techniques randomize the data, a process which violates the temporal requirement of the data. To remedy this situation, our predictive models were trained using the blocked cross-validation technique (Bergmeir and Benitez

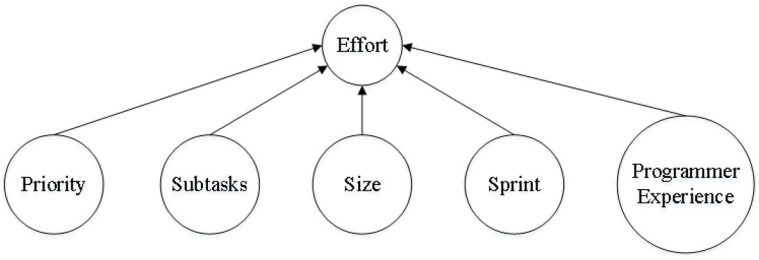


Fig. 5 Predictive model —effort per story (person-hours)

Table 5 Decision Tree for development effort

Variable	Feature importance
Sprint	0
Priority	0
Subtask	0.028
Size	0.408
Programmer 1 Experience	0.142
Programmer 2 Experience	0.421
Programmer 3 Experience	0

2011), a technique that has been considered in prior studies (Bayley and Falessi 2018; Jonsson et al. 2016), where temporal precedence is followed. Using this technique, data from sprint category 1 was used as training data while data from sprint category 2 served as the test data. Hyperparameter optimization for predictive algorithms was conducted across all the sprint categories, incrementing the sprint category for each round and appending the prior sprints to the training data. That is, hyperparameter for the predictive algorithms were chosen based on the model's predictive accuracy (means absolute error) across all the sprint categories. While sprint category 5 served as the test data, sprint categories 1-4 served as the training data. Lack of training data prevents prediction for stories in sprint category 1. Future research may include cross-project data to estimate effort required in sprint category 1 of a project. As we did not slice the dataset based on project attributes, we considered data from all the projects to train our models.

We used seven predictive algorithms identified from the prior literature on effort estimation[2] (Dejaeger et al. 2012) and machine learning (Hastie et al. 2008) to predict story-level development effort. They are (a) Bayesian network (BN), (b) ridge regression (RR), (c) artificial neural networks (ANN), (d) support vector machine (SVM), (e) decision trees (DT), (f) K-Nearest Neighbors (KNN), and (g) ordinary least squares regression (OLS). The criteria[3] for selection of these algorithms are three in number: (a) representative algorithms from prior work in effort estimation of traditional and agile software development projects, (b) newer machine learning algorithms, and (c) the ready availability of algorithms in standard machine learning libraries. To our knowledge, ridge regression, decision tree, and k-nearest neighbors have not been previously used (point (b) above) in agile effort estimation. To improve predictive accuracy, each predictive algorithm was combined with a univariate feature selection procedure (select k best features based on mutual information —measure of mutual dependence between two variables). Feature selection improved performance for support vector machine, ridge regression, and artificial neural network, while feature

[2]Dejaeger et al. (2012) identify 13 data mining techniques used in software effort estimating in a traditional setting. In this paper, we identify representative techniques (regression, decision trees, support vector machines, neural network, Bayesian network) that perform better than the variants and augment them with newer data mining techniques (k-nearest neighbor and ensemble approaches). For example, a radial kernel outperformed other kernels in Support Vector Machines.

[3]Prior effort estimation studies have considered different variants of the algorithms considered in this study. Readers are referred to studies by Jørgensen and Shepperd (2007), Wen et al. (2012), and Idri et al. (2016) for a review of candidate algorithms in traditional software development projects.

Table 6 Training and test splits for experiments

Starting sprint category for training data	Ending sprint category for training data (n)	Test data ($n+1$)
1	1	2
1	2	3
1	3	4
1	4	5
1	5	6

selection resulted in poor performance for decision tree, Bayesian network, k-nearest neighbors, and linear regression.[4]

All algorithms were implemented in *Python* programming language using the Scikit-learn (version 0.18) machine learning library (Pedregosa et al. 2011) and their hyperparameters were optimized (refer to the appendix 1 for more details). As a general strategy to tune our models, we implemented an exhaustive grid search to find possible hyperparameter values that provided optimal values. The possible range of hyperparameters to evaluate was determined based on the predictive algorithm under consideration. For example, k-nearest neighbors could not include $k > \mathrm{N}$, where N was the number of data points in the training data. Appendix A discusses ranges of values considered for each algorithm's hyperparameters and the determined optimal values. Throughout the hyperparameter tuning process, we used the blocked cross-validation technique and the Mean Absolute Error (MAE) as the score function to determine optimal hyperparameter values. Thus, each candidate's hyperparameter(s) were used to predict each story from sprint category $n+1$ while data points up to sprint category n were used as training data, where $1 \leq \mathrm{n} \leq 5$ (Table 6 provides training and corresponding test sets considered in experiments). To provide point estimates from the Bayesian network, we implemented the algorithm by Pendharkar et al. (2005), which computes the final prediction for a test instance by multiplying the group's mean with the group's predicted probability. For example, if a test story's probability for ten effort classes is 0.25, 0.75, 0.25, 0, 0, 0, 0, 0, 0, 0 and the corresponding mean of effort categories from training dataset is [10, 12, 13, 15, 16, 18, 19, 20, 22, 25], then the point estimate is $(0.25 * 10) + (0.75 * 12) + (0.25 * 13) = 2.5 + 9 + 3.25 = 14.75$ person-hours.

To evaluate the performances of the predictive algorithms, we used three metrics (Table 7) that have been used in the agile effort estimation literature (Mahnic and Hovelja 2012; Miyazaki et al. 1991; Usman et al. 2014) and/or the machine learning literature (Shmueli et al. 2016). MBE balances the overestimation and underestimation of story effort (Mahnic and Hovelja 2012; Miyazaki et al. 1991; Usman et al. 2014) to address the criticism of MRE (Mahnic and Hovelja 2012).

Tables 8 (MAE), 9 (MBE) and 10 (RMSE) show the performance of individual predictive algorithms on the three measures (Table 7) respectively across different test data sprint categories. None of the predictive algorithms consistently outperformed others across sprints on any of the three evaluation measures. Table 11 shows the aggregated performance of the individual predictive algorithms.

[4]Log Transformation of dependent variable provided worse performance.

Table 7 Evaluation measures

Measure	Formula
Mean Absolute Error (MAE)	$\frac{1}{n}\sum_{i=1}^{n}\lvert actual_i - prediction_i\rvert$
Mean Balanced Error (MBE)	$\frac{1}{n}\sum_{i=1}^{n}\frac{\lvert actual_i - prediction_i\rvert}{min(actual_i, prediction_i)}$
Root Mean Square Error (RMSE)	$\sqrt{\frac{\sum_{i=1}^{n}(actual_i - prediction_i)^2}{n}}$

Table 8 Mean Absolute Error (MAE) for individual predictive algorithms

Algorithm	Test data sprint category				
	2	3	4	5	6
Support Vector Machine (SVM)	8.81	8.41	**8.30**	7.04	**7.86**
Ridge Regression (RR)	9.84	9.07	8.62	7.06	9.34
Artificial Neural Network (ANN)	**8.59**	**7.48**	8.63	7.85	9.06
K-Nearest Neighbors (KNN)	9.64	8.34	8.59	**6.61**	8.09
Decision Tree (DT)	8.9	9.37	8.38	7.22	8.49
Linear Regression (OLS)	10.63	10.40	8.79	8.04	9.23
Bayesian Network (BN)	10.26	10.86	9.54	7.78	10.37

Bold symbols are smallest values in each column

Table 9 Mean Balanced Error (MBE) for individual predictive algorithms

Algorithm	Test data sprint category				
	2	3	4	5	6
Support Vector Machine (SVM)	2.86	2.57	2.62	3.11	4.28
Ridge Regression (RR)	3.85	2.89	2.73	3.25	5.39
Artificial Neural Network (ANN)	**2.40**	**1.84**	**1.47**	3.90	4.60
K-Nearest Neighbors (KNN)	3.55	2.54	2.51	**2.75**	3.70
Decision Tree (DT)	2.63	2.91	2.96	3.17	**3.65**
Linear Regression (OLS)	4.55	3.64	3.70	4.07	5.72
Bayesian Network (BN)	5.13	4.82	4.51	4.74	6.81

Bold symbols are smallest values in each column

Table 10 Root Mean Squared Error (RMSE) for individual predictive algorithms

Algorithm	Test data sprint category				
	2	3	4	5	6
Support Vector Machine (SVM)	13.90	12.50	13.00	10.45	13.75
Ridge Regression (RR)	13.51	12.51	**11.40**	9.37	13.66
Artificial Neural Network (ANN)	14.34	11.79	11.89	10.69	13.52
K-Nearest Neighbors (KNN)	13.06	**11.16**	12.87	**8.84**	**12.93**
Decision Tree (DT)	**13.02**	13.48	12.12	9.66	13.80
Linear Regression (OLS)	13.89	13.93	12.17	10.85	13.44
Bayesian Network (BN)	14.08	13.60	12.42	10.46	14.68

Bold symbols are smallest values in each column

Table 11 Results for individual predictive algorithms (aggregated across all test data sprint categories – 2, 3, 4, 5, 6)

Algorithm	MAE (mean absolute error)	MBE (mean balanced error)	RMSE (root mean square error)
Support Vector Machine (SVM)	**8.07**	3.07	12.76
K-Nearest Neighbors (KNN)	8.24	2.99	**11.87**
Artificial Neural Network (ANN)	8.32	**2.82**	12.49
Decision Tree (DT)	8.44	3.09	12.45
Ridge Regression (RR)	8.75	3.59	12.14
Linear Regression (OLS)	9.42	4.33	12.85
Bayesian Network (BN)	9.72	5.18	12.92

Bold symbols are smallest values in each column

In addition to using evaluation measures from prior work, we also performed statistical tests[5] to compare the performance of candidate algorithms. First, we performed the Friedman's test to determine whether there were significant differences between the errors of candidate algorithms. Friedman's test was chosen to incorporate the blocking factor introduced by the sprint category. The test results showed significant differences ($\alpha = 0.05$) between individual algorithms for mean absolute error (p-value = 0.002521) and mean balanced error (p-value = 0.000965). However, Friedman's test showed statistically insignificant differences between individual algorithms for root mean square error (p-value = 0.1235). Further, a paired Wilcoxon signed test was performed to test whether individual predictive algorithms' errors showed statistically significant difference(s). Since we were performing multiple comparisons, the probability of Type I errors increased. To account for this multiple comparison problem we used the false discovery rate (FDR) correction method by Benjamini and Hochberg (Benjamini and Hochberg 1995). For each of the evaluation measures, none of the predictor algorithms showed statistically significant difference (p-value > 0.05) when compared with other algorithms. Although we used Wilcoxon signed test to identify statistically significant differences in the performance of individual predictive algorithms, the test cannot quantify the magnitude of these differences. To quantify the differences between prediction errors for individual predictive algorithms, we used Cliff's δ (Cliff's delta), which is a non-parametric measure[6] to identify the number of times a value from a group is higher than a value from another group. We used effect size as done in Chowdhury et al. (2018) because of our limited sample size, which limited the statistical power (Neill 2008). Appendix B[7] provides effect sizes for each predictive algorithm pair with associated 95% confidence intervals (Tables 24, 25 and 26). The effect is statistically significant when the confidence interval does not contain zero, otherwise the effect is insignificant (Lee 2016). Using the effect sizes across the three evaluation

[5]We thank the anonymous reviewer for pointing this out.

[6]The measure has four categories: (a) negligible effect ($|d| < 0.147$), (b) small effect ($|d| < 0.33$), (c) medium effect ($|d| < 0.474$), and (d) large effect ($|d| > 0.474$).

[7]To facilitate practical interpretation, we also provide (Vargha and Delaney 2000) statistic ($\hat{A_{12}}$) for each pair of predictive algorithms.

measures (Tables 24, 25 and 26), we were unable to identify a predictive algorithm that was statistically significant in outperforming other remaining algorithms. However, BN and OLS showed statistically higher MAE errors in comparison to other algorithms with effect sizes being negligible or small. Thus, we removed linear regression and Bayesian network algorithms from further consideration.

Table 11 illustrates the different types of errors for individual predictive algorithms. For example, on average, the absolute error for SVM was smaller than that for KNN. However, the magnitude of error produced by SVM was greater than that of KNN. Also, different algorithms outperformed each other in different sprint categories. For example, k-nearest neighbor (KNN) performed best in sprint category 5 (Table 8) whereas support vector machine (SVM) outperformed others in sprint categories 4 and 6 (Table 8). Thus, it was not possible to know, a-priori, which algorithms were ideal for our data set, without testing. Therefore, to have a predictive algorithm that could consistently provide good predictions in a variety of situations, we developed an ensemble-based algorithm. This algorithm consistently performed well as shown by the computational results later. The superior performance is attributed to the leveraging of diversity in the performance of the component predictive algorithms in the ensemble.

6 An Ensemble Algorithm

Figure 6 provides the architecture of our predictive ensemble-based algorithm. χ represents the feature vector (prediction variables identified by the model in Fig. 5) of the story for which estimation is required. The same set of feature vectors is provided to all the predictive algorithms (P_1, P_2, ..., P_n). Each predictive algorithm provides an estimation ($\widehat{y_1}$, $\widehat{y_2}$, ..., $\widehat{y_n}$) for the new story, aided by its model which is trained using prior stories. Based on the individual predictive algorithm weights (w_1, w_2, ..., w_n), the ensemble aggregator f aggregates estimates to provide the ensemble estimation for the story ($\widehat{Y_{ensemble}}$).

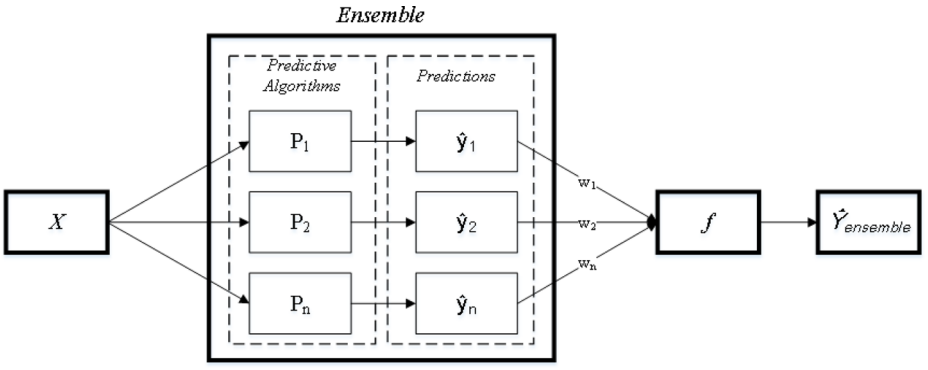


Fig. 6 Ensemble architecture

Algorithm 1 Ensemble Prediction

Initialization : W = $w_{i,1}$, set of weights; $w_{i,1} = \frac{1}{n}$ for each predictive algorithm $i \in N$, where $n = |N|$; $\beta = 1$; X = feature vector χ_k for testing; Y = actual effort value Y_k of test data | Y_k corresponds to χ_k; $Z = \phi$ (set of predictive algorithm predictions for each feature vector χ_k);
For $k = 1..|X|$
    Each predictive algorithm i provides a prediction $\widehat{y_{i,k}}$ for χ_k; $Z_{i,k} = \widehat{y_{i,k}}$; $Z = Z \cup Z_{i,k}$
    $\widehat{Y_k} = \sum_{i=1}^{n} w_{i,k} * \widehat{y_{i,k}}$
    $Error_{total,k} = \sum_{i=1}^{n} |\widehat{Y_k} - \widehat{y_{i,k}}|$
    $\beta = optimalBeta(Z,Y,n)$
    For each predictive algorithm $i \in N$
        $e_i = \frac{|Y_k - Z_{i,k}|}{Error_{total,k}}$
        $\delta_i = f_\beta(e_i)$
        $w_{i,k}^{'} = w_{i,k} * \delta_i$
    $Total = \sum_{i=1}^{n} w_{i,k}^{'}$
    For each predictive algorithm $i \in N$
        Normalize weights: $w_{i,k+1} = \frac{w_{i,k}^{'}}{Total}$
        $W = W \cup w_{i,k+1}$

FUNCTION *optimalBeta(Z,Y,n)*
    Initialization: *min* = 0.1; *max* = 8.0; *minMAE* = 0; *maxMAE* = 0; *midMAE* = 0; *stepsize* = 0.01; ϵ (tolerance) = 0.1
    Repeat steps 18 through 28 while $|max - min| > \epsilon$
        minMAE = computeMAE(Z,Y,min,n)
        maxMAE = computeMAE(Z,Y,max,n)
        mid = (min+max)/2
        midMAE = computeMAE(Z,Y,mid,n)
        If *minMAE* < *midMAE*
            max = (min+max)/2
        Else If *maxMAE* ≤ *midMAE*
            min = (min+max)/2
        Else
            min = min + stepsize
            max = max - stepsize
    Return $\beta = min$

FUNCTION *computeMAE(Z,Y,beta,n)*
Initialization : W = $w_{i,1}$, set of weights; $w_{i,1} = \frac{1}{n}$ for each predictive algorithm $i \in N$,where $n = |N|$; $\beta = beta$;
For $k = 1..|Z|$
    $\widehat{Y_k} = \sum_{i=1}^{n} w_{i,k} * Z_{i,k}$
    $Error_{total,k} = \sum_{i=1}^{n} |Y_k - Z_{i,k}|$
    For each predictive algorithm $i \in N$
        $e_i = \frac{|Y_k - Z_{i,k}|}{Error_{total,k}}$

37: $\delta_i = f_\beta(e_i)$
38: $w'_{i,k} = w_{i,k} * \delta_i$
39: $Total = \sum_{i=1}^{n} w'_{i,k}$
40: For each predictive algorithm $i \in N$
41: Normalize weights: $w_{i,k+1} = \frac{w'_{i,k}}{Total}$
42: Return $\left(\frac{1}{|Z|}\sum_{j=1}^{|Z|} |Y_j - \widehat{Y}_j|\right)$

To update weights, a reward function is used. This function is given by $f_\beta = -(e_i)^\beta + 1$, where e_i is the i^{th} predictive algorithm's contribution to the overall error and $\beta > 0$. The scaling factor for the i^{th} algorithm's weight is $\delta_i = f_\beta(e_i)$, where $0 < \delta \leq 1$. A family of reward functions for different values of parameter β in the range 0.1 to 10 is shown in Fig. 7. The weight update rule (step 9) was used for three reasons: (a) the rule was not computationally expensive, (b) the rule balanced predictive algorithm performances across past and near-past cases, and (c) the changes in the weights were gradual rather than abrupt. The ensemble algorithm is presented in Algorithm 1. To understand the reward function, consider the prediction errors contributed (Table 12) by the five algorithms across 5 stories and subsequent errors in percentages (in decimals) (step 8) (Table 13). Based on the optimal β value, weights (step 9) are updated (Table 14). Finally, the normalized weights (step 13) are computed (Table 15).

The function *optimalBeta* determines the best reward function by computing the optimal β based on MAE in the range *min* and *max*, using training data. MAE is given by $\frac{1}{|Z|}\sum_{k=1}^{|Z|} |actual_k - prediction_k|$, where Z represents the set of predictive algorithm predictions for each feature vector in the training data and is computed by the function *computeMAE*. Instead of exhaustive enumeration, a variation of binary search is used by the

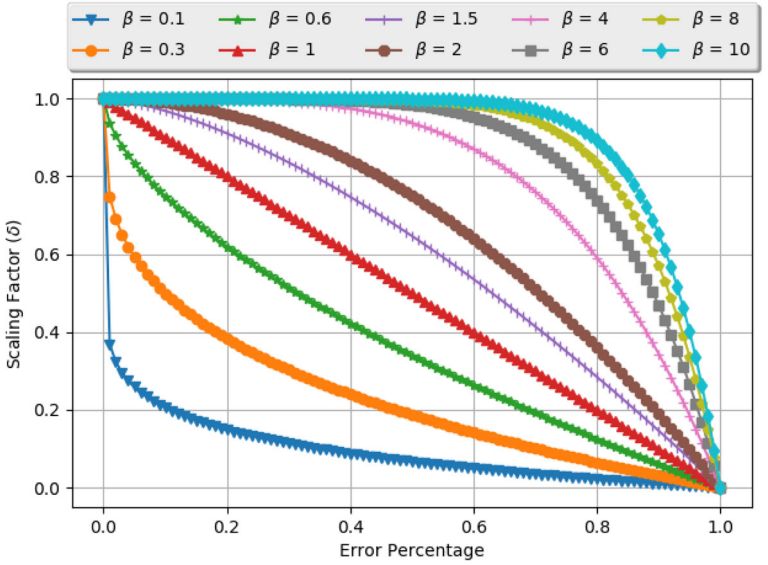


Fig. 7 Reward function

Table 12 Prediction error by individual predictive algorithms

Story number (k)	y_k	Error ($\lvert y_k - \widehat{y_{i,k}} \rvert$)					Error total ($Error_{total,k}$)
		SVM	KNN	ANN	DT	RR	
1	31.75	12.21	28.57	30.89	27.58	20.30	119.54
2	6.75	4.51	5.21	3.15	3.13	4.64	20.64
3	5.25	0.92	15.77	11.93	10.35	4.64	43.61
4	17.00	11.44	4.39	10.91	7.93	1.10	35.76
5	4.75	0.57	4.22	2.44	1.43	4.64	13.31

Table 13 Contributed error percentage (in decimals) by individual predictive algorithms

Story number (k)	y_k	Error percentage (in decimal)				
		SVM	KNN	ANN	DT	RR
1	31.75	0.10	0.239	0.258	0.231	0.170
2	6.75	0.22	0.25	0.15	0.15	0.22
3	5.25	0.02	0.36	0.27	0.24	0.11
4	17.00	0.32	0.12	0.31	0.22	0.03
5	4.75	0.04	0.32	0.18	0.11	0.35

Table 14 Updated weights of predictive algorithms

Story number (k)	β	Updated weights($w'_{i,k}$)				
		SVM	KNN	ANN	DT	RR
1	1.00	0.18	0.15	0.15	0.15	0.17
2	1.00	0.18	0.14	0.16	0.16	0.16
3	8.00	0.22	0.18	0.20	0.20	0.20
4	0.10	0.02	0.03	0.02	0.03	0.06
5	0.10	0.04	0.02	0.02	0.03	0.04

Table 15 Normalized weights

Story number (k)	Weights				
	SVM	KNN	ANN	DT	RR
Initial	0.200	0.200	0.200	0.200	0.200
1	0.224	0.190	0.185	0.192	0.208
2	0.220	0.178	0.197	0.204	0.201
3	0.220	0.178	0.197	0.204	0.201
4	0.141	0.202	0.132	0.171	0.354
5	0.254	0.146	0.137	0.228	0.236

function *optimalBeta* to determine the optimal β, starting with the initial range of β: 0.1 to 8.0[8]; the search range is reduced in half by comparing the MAE values at *min* and *max* with the MAE value at the *midpoint*. If the MAE at the *midpoint* is better than at *min* (*max*), *min* (*max*) is increased (decreased) by *stepsize*, which decreases the search space. The function converges to the optimal β, when the MAEs at *min* and *max* are less than or equal to the tolerance limit (ϵ). The function returns *min* as the optimal β for the training dataset. Thus, once the optimal β in Step 6 of EnsemblePrediction (EP) is determined, the contribution to the overall error, multiplicative scaling factor, and new non-normalized weight are determined in Step 7. Finally, in Step 12, the new weights are normalized. These normalized weights are used for ensemble prediction of effort for future stories.

7 Results

Similar to individual predictive algorithms, our ensemble-based prediction algorithm was used to predict individual story efforts in our dataset. Based on effect size analysis, SVM, ANN, KNN, DT, and RR, were retained as components of the ensemble. At the start of each blocked cross-validation dataset, we normalized the weights. Subsequent changes in the weights of individual predictors are illustrated in Fig. 8. Interestingly, the DT algorithm accrues the greatest proportion of weight in test datasets for sprint categories 2, 3, and 4. However, it receives the least weight in test datasets for categories 5 and 6. Similar differences in weights can be seen for RR, SVM, and ANN. While similar stories tend to require similar effort, there exists significant variation in the actual effort expended by the team. As we discuss later, machine learning algorithms are better positioned to learn these variations and provide better effort estimates than humans do.

Table 16 (MAE), Table 17 (MBE), and Table 18 (RMSE) provide comparisons across various ensemble-based approaches. EnsemblePrediction (EP), our ensemble-based approach, outperformed other ensemble-based approaches based on MAE and MBE (Table 19). In addition to the evaluation measures, we also performed statistical tests to compare the performance of candidate algorithms. Friedman's test was used to incorporate the blocking factor introduced by the sprint category. The test results showed significant differences ($\alpha = 0.05$) between individual algorithms for mean absolute error (p-value = 0.00075), mean balanced error (p-value = 0.0033), and root mean square error (p-value = 0.00543). Further, a paired Wilcoxon signed test was performed to test whether ensemble-based algorithmn errors were significantly different. To account for this multiple comparison problem, we used the false discovery rate (FDR) correction method by Benjamini and Hochberg (1995) which controls for false discoveries. For MAE (Table 16), MBE (Table 17), and RMSE (Table 18), the performances of all ensemble approaches were not statistically significant. Appendix C provides effect sizes for ensembles. Interestingly, the MAE and RMSE differences between our ensemble approach (EP) and other ensemble approaches (except for AVG) were lower and statistically significant. None of the other ensemble approaches show lower and statistically significant result across the evaluation measures (Table 27, Table 28, Table 29). For EP and AVG, we note that the effect size is negligible. For practical interpretation, Vargha and Delaney (2000) statistics ($\hat{A_{12}}$) compares the probability of yielding lower absolute error for EP compared to AVG (Li et al. 2017). The ($\hat{A_{12}}$) value of 0.5212

[8]Increasing β leads to higher computation costs. We choose the maximum value as 8.0 based on our experimental results. Values greater than 8.0 did not significantly improve the results.

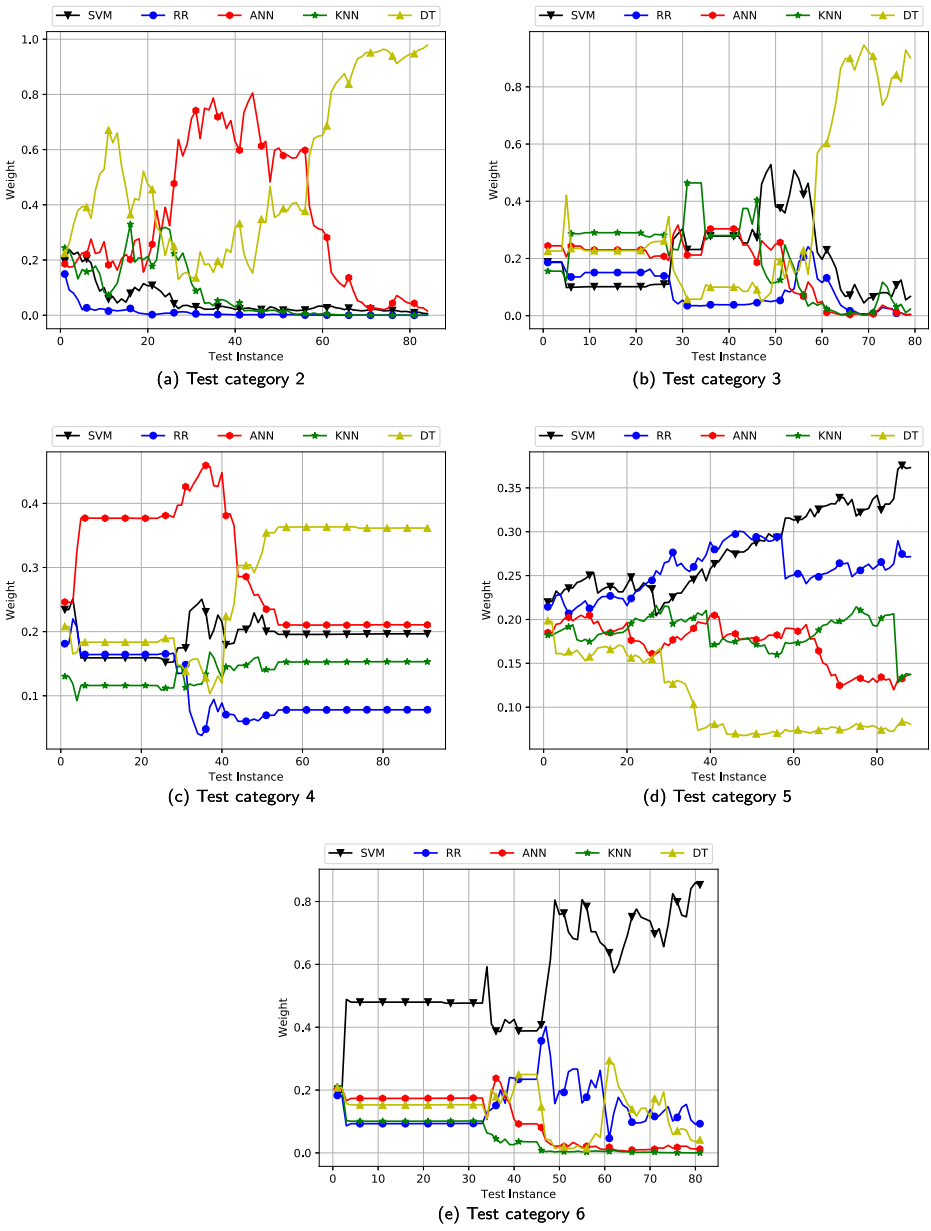


Fig. 8 Change in weights across each test category

showed that 52.12% of the time, AVG introduced higher rate of prediction errors compared to EP.

While the computation experiments did indicate that individual predictive algorithms (i.e., components of the ensemble) could occasionally provide better predictions than others within the ensemble, no individual predictive algorithm consistently outperformed others all the time. Table 20 summarizes accuracy statistics for 10 stories from 2 projects in our

Table 16 Mean Absolute Error (MAE) for ensemble-based approaches

Algorithm	Test data sprint category				
	2	3	4	5	6
EnsemblePrediction (EP)	**8.167**	**8.154**	**8.104**	**6.733**	**8.124**
Average (AVG)	8.261	8.196	8.105	6.736	8.153
Extra Trees (ET)	10.854	8.382	8.144	6.798	8.831
Random Forest (RF)	9.824	8.905	8.143	6.950	8.500
Gradient Boosting (GB)	10.476	8.832	8.135	7.074	8.908
Ada Boost (AB)	9.535	10.160	8.425	7.127	9.954
Stacking (ST)	10.037	9.678	9.324	7.939	9.036

Bold symbols are smallest values in each column

dataset for which team estimates were available. As seen from the table, EP outperformed human teams. Notwithstanding the clear differences in the errors rates for EP and human teams, our inferences from Table 20 are limited by (a) the small sample size, and (b) the lack of information pertaining to the projects such as context, processes, and constraints.

8 Practical Application

In this section, we use our predictive model and ensemble (EP) to plan each sprint for two projects from our dataset. EP provides effort estimates for stories given input features like size, sprint, developer experiences, priority, and the number of subtasks. Usman et al. (2015) report that sprint planning involves the wide use of one or more effort estimation techniques. Typically, sprint planning (a time-boxed event) involves two key decisions: (a) the selection of stories from the product backlog to be developed in the sprint, and (b) a plan that determines how the selected stories would be developed. These decisions are made by the development team, in consultation with the product owner.

To demonstrate our predictive model's utility to the development team, we focus on the first decision in sprint planning: the selection of stories from the product backlog, a decision recognized as a key application area of effort estimation (Usman et al. 2015). During sprint planning, the product backlog (maintained by the product owner) provides a one-stop access to all the stories of a project. Stories are ordered based on the decreasing order of maturity

Table 17 Mean Balanced Error (MBE) for ensemble-based approaches

Algorithm	Test data sprint category				
	2	3	4	5	6
EnsemblePrediction (EP)	**2.443**	**2.370**	2.713	**3.072**	4.232
Average (AVG)	2.856	2.521	2.691	3.096	**4.168**
Extra Trees (ET)	4.836	2.688	**2.568**	3.148	4.523
Random Forest (RF)	4.106	2.962	2.736	3.199	4.661
Gradient Boosting (GB)	3.949	3.045	2.889	3.246	4.382
Ada Boost (AB)	4.089	3.523	3.260	3.181	4.852
Stacking (ST)	3.996	3.378	2.986	3.206	4.996

Bold symbols are smallest values in each column

Table 18 Root Mean Squared Error (RMSE) for ensemble-based approaches

Algorithm	Test data sprint category				
	2	3	4	5	6
EnsemblePrediction (EP)	12.957	11.768	**11.753**	9.243	13.530
Average (AVG)	**12.300**	11.609	11.826	9.256	**12.960**
Extra Trees (ET)	15.135	**11.175**	12.075	**8.923**	13.758
Random Forest (RF)	13.596	11.912	11.962	8.993	13.263
Gradient Boosting (GB)	15.171	11.792	12.048	9.620	13.516
Ada Boost (AB)	13.680	13.896	11.918	9.623	14.471
Stacking (ST)	15.419	14.010	15.012	12.114	14.876

Bold symbols are smallest values in each column

(all components of the story are known for mature stories). A subset of product backlog consists of all mature stories for which the development team estimates effort. Given the capacity available in person-hours at each sprint, the task is to select stories from a set of mature stories such that the capacity of the development team is utilized to the maximum, while maximizing the number of story points developed. However, the selection of appropriate stories without information systems can be challenging for the following reasons: (a) team estimates can be erroneous, and (b) the optimal selection of stories would require the consideration of effort estimates across various combinations of developers.

According to agile development practices (Schwaber and Sutherland 2016), developer pairs are determined during sprint planning and any changes to the team composition is resisted during the sprint. Sprint reviews and retrospectives are dedicated activities usually conducted after the sprint to address any changes to the development procedures. The notations in Table 21 are used in the optimization model that we propose to select stories and developers for a specific sprint.

The objective function of the optimization model (Fig. 9) maximizes the number of story points to be developed in a specific sprint for which planning is underway. Constraint (1) ensures that every story, if selected for development, is developed by a specific developer or a developer pair. Constraint (2) ensures that the total effort predicted across a pair of developers for selected stories is less than or equal to the available capacity. If developer A and B develop a story as a pair, then the effort required is counted only once. Constraint

Table 19 Results for ensemble-based approaches (Aggregated across all test data sprint categories – 2, 3, 4, 5, 6)

Algorithm	MAE (mean absolute error)	MBE (mean balanced error)	RMSE (root mean square error)
EnsemblePrediction (EP)	**7.845**	**2.961**	11.906
Average (AVG)	7.877	3.059	**11.631**
Extra Trees (ET)	8.578	3.536	12.369
Random Forest (RF)	8.508	3.515	12.021
Gradient Boosting (GB)	8.657	3.489	12.535
Ada Boost (AB)	8.992	3.762	12.773
Stacking (ST)	9.188	3.690	14.327

Bold symbols are smallest values in each column

Table 20 Comparison of estimation accuracy for human and ensemble (EP) approaches

Estimation	MAE (mean absolute error)	MBE (mean balanced error)	RMSE (root mean square error)
Team Estimate	12.19	5.34	21.37
EnsemblePrediction (EP) Estimate	**8.243**	**1.132**	**15.331**

Bold symbols are smallest values in each column

(3) ensures that the total effort expended by each developer across all selected stories is less than or equal to her available capacity for a sprint. Constraint (4) ensures that the order of developer pair does not matter. Constraint (5) indicates that the decision variable is a binary 0/1 integer variable.

To instantiate this optimization model, we needed (a) data pertaining to actual effort expended by each developer for the project and (b) any other effort expended by the same set of developers on other concurrent projects. Effort expended on other concurrent projects allowed us to estimate available effort for each developer. Note that the effort expended on the selected project was already known to us from our dataset. By accessing the project management tool of the university providing data, we could identify two projects from our dataset such that the total effort expended (for the selected project and other concurrent projects) by the developers involved in these projects could be estimated.

The two selected projects had 43 and 29 stories respectively. We refer to these projects as DEMO-1 and DEMO-2. Using the other 22 projects as training data, the predictive models were trained and the efforts for all the stories of DEMO-1 and DEMO-2 were separately predicted (DEMO-1 data was used for the training model when predicting DEMO-2 stories) using EP. For each story in DEMO-1 and DEMO-2, the developers were represented individually and in pairs to determine effort estimates in each of the cases. For sprint planning, we assumed that the stories from the current sprint and the immediate next sprint were mature. Thus, in Table 22, we have stories from sprints 5 and 6 (these are not categories of the sprint but the actual sprint numbers from the data) of DEMO-1. The dataset indicated

Table 21 Notations used in the model

Notation	Description
S	Index set of mature stories to be developed
$x_{i,j,k}$	{= 1, if developers i and j assigned to develop story $k \in S$, 0 otherwise}
n	$n = \|S\|$; Total number of mature stories
m	Total number of stories selected for development (from set S) in the current sprint
P	Index set of developers; $P = \{0, ..., p\} \| i \in P, i = 0 \rightarrow$ no developer
$c_{i,j,k}$	Total Estimated Effort (in hours) required for developers i and j to develop story k. This estimate is provided by the ensemble (EP), using story variables and experience of developers; $i, j \in P$; $j = 0$ when only one developer develops the story; We assume $c_{i,j,k} = c_{j,i,k}$
e_i	Total effort (in hours) developer i can expend on the current project in the current sprint (assumed to be 40 hours per week)
$size_k$	Size (story points) of story k

$$\text{Maximize Z} = \sum_{i\in P}\sum_{\substack{j\in P\\ j>i}}\sum_{k\in S} size_k * x_{i,j,k}$$

$$\text{subject to} \quad \sum_{i\in P}\sum_{\substack{j\in P\\ j\neq i}} x_{i,j,k} \leq 1, \qquad \forall i \forall j \in P \quad \forall k \in S \tag{1}$$

$$\left[\left(\sum_{\substack{g\in P\\ g\neq i}}\sum_{k\in S} c_{g,i,k} * x_{i,g,k} + \sum_{\substack{h\in P\\ h\neq j}}\sum_{k\in S} c_{h,j,k} * x_{h,j,k}\right) - \sum_{k\in S} c_{i,j,k} * x_{i,j,k}\right] \leq (e_i + e_j), \quad (i,j) \in \{P \times P | i < j\} \tag{2}$$

$$\sum_{i\in P}\sum_{\substack{j\in P\\ j\neq i}}\sum_{k\in S} c_{i,j,k} * x_{i,j,k} \leq e_i, \quad \forall i \forall j \in P \quad \forall k \in S \tag{3}$$

$$x_{i,j,k} = x_{j,i,k}, \quad \forall i \forall j \in P, \quad \forall k \in S, \tag{4}$$

$$x_{i,j,k} = 0 \quad or \quad 1 \quad integer, \quad \forall i \forall j \in P, \quad \forall k \in S \tag{5}$$

Fig. 9 Optimization model

that one story (A) was developed in sprint 5, while 3 stories (B, C, D) were developed in sprint 6. Each story had the same size (8 story points). Finally, the predicted developer efforts for the cases when developers worked individually or in pairs were used as model parameters. It can be seen that the predicted efforts for Developer 1 and Developer 2 were the same across the 4 stories because they had the same years of experience. While sprint lengths could vary, typically from 1-3 weeks, our dataset had sprints that were 2 weeks long.

Using the project management tool of the university, we retrieved the efforts expended by Developer 1 and Developer 2 on other concurrent projects during the two weeks in which sprint 5 for DEMO-1 was active. Table 23 provides the total effort expended by the developers across other concurrent projects and the available capacity for DEMO-1. The total available effort for a developer was 40 hours per week, which was the standard university assignment. Applying our optimization model to sprint 5, the model allocated stories A, B, and C to Developer 1 such that the total predicted effort to be expended was 41.86 hours which was less than the available capacity for Developer 1 (43.5 hours —see Table 23). Developer 2 did not have adequate capacity in sprint 5 to handle any of the stories. Thus, story D was not assigned to any developers in sprint 5. Our model was implemented using IBM ILOG CPLEX Optimization Studio (version 12.7).

Using the optimization model, we planned stories for each sprint for projects DEMO-1 and DEMO-2. For DEMO-1, our sprint planning scheduled the project completion in 11 sprints. Based on the data available from the university, the actual project took 16 sprints to complete. Also, using our sprint planning optimization model, the unused available effort could be reduced by 323.97 hours (76.58%) in comparison to the actual project. For DEMO-2, our sprint planning scheduled the project completion in 7 sprints, whereas the actual project took 14 sprints. Further, the unused available effort could be reduced by 247.4

Table 22 Sprint planning (DEMO-1)

Actual Sprint Number (from data)	5	6	6	6
Story ID	A	B	C	D
Actual Story Size	8	8	8	8
Predicted Developer 1 Effort (hours)	12.53	15.42	13.91	14.84
Predicted Developer 2 Effort (hours)	12.53	15.42	13.91	14.84
Predicted Developer1-2 Effort (hours)	22.67	22.34	20.33	24.46

Table 23 Available capacity for Sprint 5 of project DEMO-1

Developer	Effort	
	Total expended effort across other projects (E)	Capacity available
Developer 1	36.5	43.5 (80-E)
Developer 2	71.25	8.75 (80-E)
Developer 1-2	107.75	52.25 (160-E)

hours (49.93%) in comparison to the actual project. Note that our sprint planning optimization model, when instantiated with data, covered only two consecutive sprints at a time. However, the model instance can be enhanced to cover a greater number of sprints.

In order to assess the utility of our approach to practice, we conducted three informal discussions with developers/project managers. During the discussions, the following points emerged. First, the effort estimation model was useful for sprint planning as it could lead to the completion of the project on time and within budget. Second, the optimization model could be used for optimal scheduling of developers and stories to sprints. Third, there was a gradual decrease in the effort required for similar tasks over time due to learning effects. Fourth, notwithstanding the benefits of our approach, there could be situations where our approach could be "off the mark" in predicting effort, such as when developers worked on projects that required new technologies that they had not previously worked on.

9 Threats to Validity

Given the empirical nature of this study, the relevant validity checks are internal validity and external validity (Chari and Agrawal 2018; Cinnéide et al. 2017). A study has high internal validity when confounding explanations are accounted for and biases are minimized. Usman et al. (2014) report that experience and task size play critical roles in determining effort estimation. Since we include developer experience and story size in our model, we believe that we have accounted for this explanation. In addition, we augment our model with other attributes such as priority and subtasks to improve predictions. Other confounding effects include the application domain of projects and seasonal variations that may impact the effort expended by project teams. While our dataset had projects from different application domains, the superior estimates of our ensemble-based approach compared to team estimates give us the confidence that the internal validity was high.

External validity refers to the degree to which conclusions of this study can be generalized. Individual algorithms and ensembles were optimized for the dataset used in this study. Other datasets on agile software development may require additional tuning of various algorithms to identify hyperparameter values. With hyperparameter tuning and training, we believe we will reach similar conclusions for other datasets. Thus, we believe our approach is generalizable across different organizations.

10 Conclusion

In this paper, we developed a predictive model to estimate the effort required to develop stories for agile software development projects. Using a dataset of 503 stories, we employed

machine learning algorithms identified from our literature review. Since none of the predictive algorithms consistently outperformed others, we developed an ensemble-based algorithm that performed better than other ensemble-based approaches. Finally, we demonstrated the utility of our predictive model and the ensemble-based algorithm to optimize sprint planning for two projects from our dataset.

10.1 Managerial Implications

This research has significant implications for managers of agile software development projects. First, this research identifies various predictor variables that managers could track for effort prediction of stories in agile software development projects. Managers could thus set up data collection processes accordingly for project management. Second, this research informs managers that they could do better in terms of effort prediction by relying on machine learning approaches instead of relying on development team estimates. Third, this research makes available to managers a rigorous scientific approach for scheduling stories and developers to sprints, thereby leading to a reduced number of sprints. Finally, our ensemble-based approach identifies better performing algorithms for each sprint category. This significantly reduces the amount of effort required to retrain and identify candidate algorithms. Despite this advantage, project managers need to consider the skills required to use our ensemble-based approach and the optimization model.

An overarching implication from this study is that practice could benefit from embracing rigorous scientific approaches that harness the power of data analytics and optimization in planning sprints. It would thus make sense to embed machine learning and optimization modules into project management tools for efficient planning and management of agile software development projects, thus leading to efficient allocation of organization resources across different projects.

10.2 Research Implications

This research contributes to the literature on agile software effort prediction in multiple ways. First, by identifying additional predictor variables such as priority and subtask that were not previously used in effort prediction of stories. Second, by demonstrating the utility of temporal sequencing in effort estimation by implicitly accounting for learning effects. Third, by testing predictive algorithms such as ridge regression, decision tree, and k-nearest neighbors that were not previously reported in the literature for story effort estimation. Fourth, by proposing a new ensemble-based algorithm that demonstrates superior and consistent performance over existing approaches. The ensemble-based algorithm could have broader impacts beyond the software engineering literature. Fifth, by proposing an innovative optimization model that could provide a new approach for sprint planning.

10.3 Limitations

The research presented in this paper has certain limitations for the following reasons. We used existing data from a project management system. Inherent in such archival datasets are issues pertaining to bias and errors in reporting data that may propagate in our analysis. The distributions of size and priority in the data show skews that could bias the models trained in our experiments. Other datasets, depending on the application context, may have different distributions for these variables. To address these issues, we performed preliminary analysis

of the data using decision trees to identify predictive variables and their importance. Our dataset was extracted from a project management tool. This archival dataset could contain overestimation and/or underestimation of effort reported by team members (Magazinius et al. 2012). Machine learning algorithms train on this data and inherit the bias and contextual estimation techniques from the data. Thus, any bias and/or distortion may be replicated by machine learning algorithms.

While on an aggregate basis our ensemble algorithm performed better than other ensemble approaches, on an individual prediction basis, some predictive algorithms in the ensemble provided better predictions than our ensemble algorithm. In addition, our experiments were limited to the dataset from a specific organization. This has the potential of impacting the generalizability of our findings. This is an inherent limitation. Furthermore, we draw limited conclusions on the superior performance of the prediction methods presented in this study and restrict the inferences to effort estimation in agile software development projects.

During the informal discussions with developers/project managers, it emerged that in situations where developers worked on projects that required new emerging technologies that they had not previously worked on, predicting effort using models calibrated using project data pertaining to older technologies was prone to errors. This is a limitation.

Finally, while our optimization approach provided scientific rigor in scheduling stories and developers to sprints, there were limitations in terms of not fully accounting for team dynamics and the time needed for exploration and experimentation. Vidgen and Wang (2009) discuss allocation of study time for project and non-project investigations in agile approaches. Prediction and optimization steps required explicit identification, description, and the quantification of these activities which may be difficult, given the creative nature of the process.

10.4 Future Research

We see two research directions in the future. First, future research in effort estimation could include human experts in ensembles. Human expert estimates, especially when the development team transitions to working with new emerging technologies or working in new application domains, could leverage the use of context specific information which may not be accounted for by predictive algorithms.

Second, future research could consider developing efficient optimization approaches at the project portfolio level, as developers are often working on multiple projects during a given sprint period. Optimization at a portfolio level using granular data will ensure the efficient use of resources and productivity gains in software development organizations.

Acknowledgements This paper has benefited from the feedback received at Workshop in Information Technology and Systems (WITS) 2014 (Auckland) and WITS 2016 (Dublin) where preliminary versions of the paper were presented. There were many individuals that helped us with this research project. First, we would like to thank those individuals at our data site that helped us gain access to the dataset. We would also like to thank Dr. Terry Sincich for helping us with the design and choice of statistical tests, developers/project managers who shared their insights on the implications of this research to practice, and Dr. Patricia Nickinson for proofreading and editing our draft. Finally, we would like to express our sincere gratitude to the three anonymous reviewers and the editors for providing constructive feedback on our earlier submission.

Appendix A

The K-Nearest Neighbor (KNN) algorithm uses k closest data points in the training data (determined by some distance metric —in our case *Euclidean*), where k is provided by the user, to determine its prediction. Each neighbor of a target point can equally influence the prediction (uniform weight) or inverse of the distance from the target point (distance-based weight). For our experiments, the parameter optimized was the number of neighbors (k). Exhaustive searches from 2 neighbors to the maximum possible in the training dataset were used; the best number of neighbors to minimize prediction errors is 14. Further, we used a distance-based measure such that closer points had higher influence than those that were farther out.

Decision trees (DT) represent a set of hierarchical decisions on the feature variables of the training data. A decision at a particular node, called a *split criterion*, is conditional on one or more feature variables. Different criteria can be used to measure the quality of the split. In our case, mean absolute error was used as the measurement criterion. As the depth of a tree increases, the tree is over fitted to the training data i.e., all instances can be classified by a dedicated leaf node of their own. However, such trees suffer when exposed to test datasets. To balance overfitting, optimization for the *maximum depth* of the tree was found to be 2. We considered depth value from 1 to 199.

Ridge regression (RR), in addition to minimizing the residual sum of squares (RSS), aims to minimize the *shrinkage penalty* using a *tuning parameter*. As the tuning parameter approaches zero, the ridge regression produces a least squares model. As the tuning parameter increases, the flexibility of the ridge regression fit decreases, leading to decreased variances but increased bias. The optimal value of the tuning parameter was found to be 3.51. The tuning parameter value varied from 0.0001 to 50 with increments of 0.001.

A Bayesian network (BN) learns the joint probability distribution of the target variable using a causal model that provides the prior and posterior probabilities of variables. New information can be incorporated into the model using belief updating procedures. Bayesian networks are popular in machine learning and software effort prediction literature due to their accuracy (Hearty et al. 2009). We implemented the procedure in Pendharkar et al. (2005) that provided point predictions for a target story.

Support Vector Machine (SVM) is a generalization of *maximal margin classifier* that identifies a *separating hyperplane* (p-1 dimensional flat space in a p dimensional space) that classifies the one-dimensional space. However, this approach is limited by cases where the perfect separating hyperplane is not available. Support Vector Classifiers address this problem by identifying soft hyperplanes that almost separate the classes. Data points that lie on the hyperplane are known as *support vectors*. These points determine the support vector classifier performance. Tolerance for points on the wrong side of the hyperplane can be tuned by a penalty parameter C. A greater value of C will increase the tolerance for data points on the wrong side. The hyperplane can be linear or nonlinear. Support Vector Regression (SVR), a nonparametric method, uses kernel functions to identify the decision boundary. Kernel functions can be linear, polynomial, radial, or sigmoid, among others. In our experiments, the penalty parameter C was identified to be 1.9 using a Radial Basis Function (rbf) kernel whose width was identified to 0.3. We used a multi-level grid search to tune parameters. The possible values for penalty parameters were varied from 0.01 to 10 whereas the values for kernel widths varied from 0.1 to 10.

In Artificial Neural Network (ANN), derived features are created from non-linear combinations of input variables (input nodes). The target variable to be predicted, in turn, is modeled as a linear function of the derived features. The units computing the derived

features are known as hidden nodes since they are not visible to the user. Hidden nodes transform the nodes' inputs using a weighted linear summation. A regularization parameter is used to limit overfitting of the neural network to the training data. In our experiments, the regularization parameter was determined to be 0.000081. The value of regularization parameter varied from 1.00E-06 to 1 with increments of 1.00E-05.

In Random Forest (RF), a number of decision trees are applied to subsamples of the training dataset. Every time these trees are built, a different set of predictors is chosen as split candidates. This produces *decorrelated* trees which reduce the overall variance. For our experiments, the random forest was optimized to 5 trees in the forest with 4 features selected to determine the best split. Possible values for the number of trees in the forest and features were varied from 1 to 20 each, with all possible combinations considered to identify optimal parameter values.

Extra Trees (ET), similar to RF, fit multiple randomized decision trees on subsamples of the dataset. However, variables are chosen from the entire dataset rather than bootstrapping and at random. The number of trees was optimized to 5 trees and 4 features selected to determine the best split. Possible values for the number of trees and features varied from 1 to 20 each, with all possible combinations considered to identify optimal parameter values.

Adaptive Boost (AB) works by incrementally focusing on cases that are difficult to predict. A base estimator (in our case, decision tree) is identified by fitting to the original dataset. Incrementally, adjusted estimators are fitted to the dataset such that weight is increased for wrongfully classified data points. In our experiments, the boosting was terminated after two estimators were fit to the data. Possible values considered were from one to 30 estimators.

Gradient Boosting (GB) optimizes least squares loss function by adding weak learners (decision trees) to an additive model. As new trees are added at each boosting stage, existing trees are not changed. The optimal number of boosting stages was identified as 28. The possible range of stages considered was 1-60.

Stacking (ST) is an ensemble approach which aggregates individual 1^{st}-level algorithms with a 2^{nd}- level algorithm (often known as the meta-regressor). Individual first-order algorithms provide predictions which form the input to the meta-regressor. Five algorithms selected in our experiments formed the first order algorithms with respective hyperparameters identified in the experiments. We used Support Vector Machine with "rbf" kernel as the meta-regressor.

For an extensive discussion on each of these algorithms, readers are referred to (Hastie et al. 2008), (James et al. 2015), (Aggarwal 2015), and (Shmueli et al. 2016).

Appendix B

Tables in this appendix (Tables 24, 25 and 26) provide effect sizes with Cliff's delta, 95% confidence interval, and the Vargha and Delaney (2000) ($\hat{A_{12}}$) statistic (VDA), which compares the probability of yielding lower absolute error for individual predictive algorithms. For each effect size computation, we consider the error values across all test dataset (N = 423). Reversing the order of algorithms will provide the same Cliff's delta multiplied by -1. Similarly, confidence interval values will reverse with different signs. The VDA statistic for the reverse order of the algorithms is $VDA_{Reverse} = |1 - VDA_{originalorder}|$. For example, the effect size of RR and SVM is 0.10752, confidence interval is [0.02946, 0.18394], and VDA statistic is 0.553759. Statistically significant pairs are highlighted.

Table 24 Effect Sizes for Individual Predictive Algorithms (MAE)

Algorithm	Pair	Cliff's Delta	95% Confidence Interval	VDA statistic ($\hat{A_{12}}$)
SVM	RR	−0.10752	[−0.18427, −0.02946]	0.446241
SVM	ANN	−0.0221	[−0.09995, 0.056027]	0.488951
SVM	KNN	−0.07146	[−0.14859, 0.006525]	0.464268
SVM	DT	−0.04206	[−0.11962, 0.036023]	0.478972
SVM	BN	−0.21018	[−0.28481, −0.13302]	0.394911
SVM	OLS	−0.1347	[−0.21128, −0.05646]	0.432652
RR	ANN	0.07736	[−0.00065, 0.154433]	0.53868
RR	KNN	0.042749	[−0.03533, 0.120306]	0.521374
RR	DT	0.059683	[−0.01844, 0.137083]	0.529841
RR	BN	−0.09132	[−0.16811, −0.01342]	0.454342
RR	OLS	−0.0351	[−0.11269, 0.04292]	0.482451
ANN	KNN	−0.04289	[−0.12061, 0.035348]	0.478553
ANN	DT	−0.02236	[−0.10004, 0.055586]	0.48882
ANN	BN	−0.16783	[−0.24351, −0.09013]	0.416084
ANN	OLS	−0.10828	[−0.18478, −0.03048]	0.445858
KNN	DT	0.025524	[−0.05227, 0.103007]	0.512762
KNN	BN	−0.14339	[−0.21936, −0.0657]	0.428304
KNN	OLS	−0.0757	[−0.15311, 0.00263]	0.46215
DT	BN	−0.1575	[−0.23343, −0.07965]	0.421251
DT	OLS	−0.09051	[−0.16752, −0.01239]	0.454747
BN	OLS	0.048544	[−0.0298, 0.126295]	0.524272

Table 25 Effect Sizes for Individual Predictive Algorithms (MBE)

Algorithm	Pair	Cliff's Delta	95% Confidence Interval	VDA statistic ($\hat{A_{12}}$)
SVM	RR	0.01381	[−0.06383, 0.091288]	0.506905
SVM	ANN	0.0075	[−0.07033, 0.085237]	0.50375
SVM	KNN	−0.01966	[−0.09726, 0.058189]	0.490172
SVM	DT	−0.02093	[−0.09854, 0.056934]	0.489535
SVM	BN	−0.06207	[−0.13924, 0.015835]	0.468963
SVM	OLS	−0.02101	[−0.0984, 0.05664]	0.489496
RR	ANN	−0.00586	[−0.0836, 0.071957]	0.497071
RR	KNN	−0.03214	[−0.10976, 0.045883]	0.483932
RR	DT	−0.03371	[−0.11127, 0.044263]	0.483147
RR	BN	−0.07457	[−0.15145, 0.003212]	0.462717
RR	OLS	−0.03522	[−0.11259, 0.042575]	0.48239
ANN	KNN	−0.02847	[−0.10606, 0.049463]	0.485765
ANN	DT	−0.02858	[−0.10612, 0.049319]	0.485712
ANN	BN	−0.07121	[−0.14818, 0.006611]	0.464394
ANN	OLS	−0.03022	[−0.10749, 0.04742]	0.484891
KNN	DT	−0.00137	[−0.07914, 0.076407]	0.499313

Table 25 (continued)

Algorithm	Pair	Cliff's Delta	95% Confidence Interval	VDA statistic ($\hat{A_{12}}$)
KNN	BN	−0.04446	[−0.12138, 0.032978]	0.477768
KNN	OLS	−0.00277	[−0.08005, 0.074538]	0.498614
DT	BN	−0.04121	[−0.11782, 0.035899]	0.479397
DT	OLS	−0.00104	[−0.07813, 0.076064]	0.49948
BN	OLS	0.038607	[−0.03932, 0.116064]	0.519304

Table 26 Effect Sizes for Individual Predictive Algorithms (RMSE)

Algorithm	Pair	Cliff's Delta	95% Confidence Interval	VDA statistic ($\hat{A_{12}}$)
SVM	RR	−0.10742	[−0.18417, −0.02936]	0.446292
SVM	ANN	−0.02206	[−0.09992, 0.056066]	0.48897
SVM	KNN	−0.07145	[−0.14857, 0.006542]	0.464276
SVM	DT	−0.0422	[−0.11976, 0.035882]	0.478902
SVM	BN	−0.21012	[−0.28475, −0.13295]	0.394942
SVM	OLS	−0.13483	[−0.21141, −0.0566]	0.432585
RR	ANN	0.077344	[−0.00067, 0.154417]	0.538672
RR	KNN	0.042749	[−0.03533, 0.120306]	0.521374
RR	DT	0.059716	[−0.01841, 0.137115]	0.529858
RR	BN	−0.09131	[−0.16811, −0.01341]	0.454345
RR	OLS	−0.03508	[−0.11267, 0.042936]	0.48246
ANN	KNN	−0.04279	[−0.12051, 0.035453]	0.478606
ANN	DT	−0.02238	[−0.10005, 0.055568]	0.488811
ANN	BN	−0.16771	[−0.24339, −0.09]	0.416146
ANN	OLS	−0.10833	[−0.18482, −0.03053]	0.445836
KNN	DT	0.025519	[−0.05227, 0.103001]	0.512759
KNN	BN	−0.14336	[−0.21932, −0.06567]	0.428321
KNN	OLS	−0.07571	[−0.15312, 0.002619]	0.462144
DT	BN	−0.15744	[−0.23338, −0.07959]	0.421279
DT	OLS	−0.0938	[−0.17076, −0.0157]	0.453099
BN	OLS	0.048489	[−0.02986, 0.12624]	0.524244

Appendix C

Tables in this appendix (Tables 27, 28 and 29) provide effect sizes with Cliff's delta, 95% confidence interval, and the Vargha and Delaney (2000) ($\hat{A_{12}}$) statistic (VDA) for ensemble algorithms. For each effect size computation, we consider the error values across all test dataset (N = 423). Statistically significant pairs are highlighted.

Table 27 Effect Sizes for Ensemble Algorithms (MAE)

Algorithm	Pair	Cliff's Delta	95% Confidence Interval	VDA statistic ($\hat{A_{12}}$)
AVG	EP	0.043168	[−0.03465, 0.120462]	0.521584
AVG	ADBO	−0.07518	[−0.15212, 0.002674]	0.462412
AVG	GB	−0.05651	[−0.13377, 0.021428]	0.471743
AVG	ET	−0.04463	[−0.12202, 0.033301]	0.477687
AVG	RF	−0.06457	[−0.14177, 0.013414]	0.467716
AVG	ST	−0.05685	[−0.13348, 0.020451]	0.471573
EP	ADBO	−0.11421	[−0.19057, −0.03648]	0.442894
EP	GB	−0.09699	[−0.17377, −0.01905]	0.451503
EP	ET	−0.08555	[−0.16246, −0.00762]	0.457223
EP	RF	−0.10433	[−0.18107, −0.02632]	0.447837
EP	ST	−0.09858	[−0.17529, −0.02068]	0.450709
ADBO	GB	0.018873	[−0.05901, 0.09653]	0.509437
ADBO	ET	0.029928	[−0.048, 0.107495]	0.514964
ADBO	RF	0.013648	[−0.06425, 0.091378]	0.506824
ADBO	ST	0.018214	[−0.05961, 0.095819]	0.509107
GB	ET	0.011798	[−0.0659, 0.089349]	0.505899
GB	RF	−0.00848	[−0.08605, 0.069197]	0.495761
GB	ST	−0.00121	[−0.07802, 0.075604]	0.499394
ET	RF	−0.01868	[−0.09635, 0.059217]	0.490661
ET	ST	−0.01366	[−0.09051, 0.063357]	0.49317
RF	ST	0.007349	[−0.06945, 0.084065]	0.503675

Table 28 Effect Sizes for Ensemble Algorithms (MBE)

Algorithm	Pair	Cliff's Delta	95% Confidence Interval	VDA statistic ($\hat{A_{12}}$)
AVG	ENS	−7.82E-05	[−0.077702039, 0.077546496]	0.499960878
AVG	ADBO	−0.043799496	[−0.121226132, 0.034156895]	0.478100252
AVG	GB	−0.038216276	[−0.115773121, 0.039803739]	0.480891862
AVG	ET	−0.029581566	[−0.10707979, 0.048273941]	0.485209217
AVG	RF	−0.034376764	[−0.111961116, 0.043624153]	0.482811618
AVG	ST	−0.099564632	[−0.176408424, −0.021514572]	0.450217684
EP	ADBO	−0.043749197	[−0.121191882, 0.034222849]	0.478125402
EP	GB	−0.039345215	[−0.116883712, 0.038670033]	0.480327392
EP	ET	−0.029330069	[−0.106836214, 0.048530374]	0.485334965
EP	RF	−0.035036243	[−0.112622718, 0.042974874]	0.482481878
EP	ST	−0.099190182	[−0.176008171, −0.021171436]	0.450404909
ADBO	GB	0.007248685	[−0.070552037, 0.08496175]	0.503624343
ADBO	ET	0.015576011	[−0.062137906, 0.093102195]	0.507788005
ADBO	RF	0.009528919	[−0.068282183, 0.087224795]	0.50476446
ADBO	ST	−0.050684909	[−0.128071633, 0.027315278]	0.474657546

Table 28 (continued)

Algorithm	Pair	Cliff's Delta	95% Confidence Interval	VDA statistic ($\hat{A_{12}}$)
GB	ET	0.010288997	[−0.067239374, 0.087693865]	0.505144499
GB	RF	0.003280631	[−0.074209517, 0.080731399]	0.501640315
GB	ST	−0.057492078	[−0.134639177, 0.020347805]	0.471253961
ET	RF	−0.006740104	[−0.084461018, 0.071062328]	0.496629948
ET	ST	−0.067786664	[−0.144935784, 0.01018172]	0.466106668
RF	ST	−0.060258538	[−0.137392781, 0.017602135]	0.469870731

Table 29 Effect Sizes for Ensemble Algorithms (RMSE)

Algorithm	Pair	Cliff's Delta	95% Confidence Interval	VDA statistic ($\hat{A_{12}}$)
AVG	EP	0.024049	[−0.05354, 0.101345]	0.512024
AVG	ADBO	−0.07519	[−0.15213, 0.002657]	0.462404
AVG	GB	−0.05651	[−0.13377, 0.021434]	0.471746
AVG	ET	−0.04462	[−0.12201, 0.033307]	0.477689
AVG	RF	−0.0646	[−0.1418, 0.01338]	0.467699
AVG	ST	−0.05685	[−0.13348, 0.020456]	0.471575
EP	ADBO	−0.09771	[−0.17433, −0.01993]	0.451143
EP	GB	−0.07951	[−0.15651, −0.00155]	0.460244
EP	ET	−0.06689	[−0.14403, 0.011052]	0.466554
EP	RF	−0.08725	[−0.1642, −0.00925]	0.456374
EP	ST	−0.08023	[−0.15677, −0.00273]	0.459886
ADBO	GB	0.018879	[−0.05901, 0.096536]	0.509439
ADBO	ET	0.029922	[−0.04801, 0.10749]	0.514961
ADBO	RF	0.01367	[−0.06423, 0.0914]	0.506835
ADBO	ST	0.01822	[−0.05961, 0.095825]	0.50911
GB	ET	0.011792	[−0.0659, 0.089344]	0.505896
GB	RF	−0.00848	[−0.08606, 0.069191]	0.495758
GB	ST	−0.00124	[−0.07804, 0.075582]	0.499382
ET	RF	−0.01867	[−0.09634, 0.059229]	0.490667
ET	ST	−0.01368	[−0.09053, 0.063341]	0.493162
RF	ST	0.007349	[−0.06945, 0.084065]	0.503675

References

Abrahamsson P, Salo O, Ronkainen J, Warsta J (2002) Agile software development methods: Review and analysis. Report, VTT

Abrahamsson P, Moser R, Pedrycz W, Sillitti A, Succi G (2007) Effort prediction in iterative software development processes - incremental versus global prediction models. Empirical Software Engineering and Measurement, pp 344–353

Abrahamsson P, Fronza I, Moser R, Vlasenko J, Pedrycz W (2011) Predicting development effort from user stories. In: International Symposium on Empirical Software Engineering and Measurement, pp 400–403
Aggarwal C (2015) Data Mining: The Textbook. Springer, New York
Azhar D, Riddle P, Mendes E, Mittas N, Angelis l (2013) Using ensembles for web effort estimation. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement
Azzeh M, Nassif AB, Minku L (2015) An empirical evaluation of ensemble adjustment methods for analogy-based effort estimation. The Journal of Systems and Software 103:36–52
Bayley S, Falessi D (2018) Optimizing prediction intervals by tuning random forest via meta-validation. arXiv:1801.07194
Beck K, Andres C (2004) Extreme Programming Explained:Embrace Change. Addison-Wesley, Reading
Benjamini Y, Hochberg Y (1995) Controlling the false discovery rate: a practical and powerful approach to multiple testing. J R Stat Soc Ser B (Methodological) 57(1):289–300
Bergmeir C, Benitez JM (2011) Forecaster performance evaluation with cross-validation and variants. In: 11Th international conference on intelligent systems design and applications (ISDA). IEEE, pp 849-854
Chari K, Agrawal M (2018) Impact of incorrect and new requirements on waterfall software project outcomes. Empir Softw Eng 23(1):165–185
Chowdhury S, Di Nardo S, Hindle A, Jiang ZMJ (2018) An exploratory study on assessing the energy impact of logging on android applications. Empir Softw Eng 23(3):1422–1456
Cinnéide MÓ, Moghadam IH, Harman M, Counsell S, Tratt L (2017) An experimental search-based approach to cohesion metric evaluation. Empir Softw Eng 22(1):292–329
Conboy K (2009) Agility from first principles: Reconstructing the concept of agility in information systems development. Inf Syst Res 20(3):329–354
Dejaeger K, Verbeke W, Martens D, Baesens B (2012) Data mining techniques for software effort estimation: A comparative study. IEEE Trans Softw Eng 38(2):375–97
Grenning J (2002) Planning poker or how to avoid analysis paralysis while release planning. Report, Hawthorn Woods: Renaissance Software Consulting
Hastie T, Tibshirani R, Friedman J (2008) The Elements of Statistical Learning. Springer, New York
Haugen NC (2006) An empirical study of using planning poker for user story estimation. In: Agile Conference, 2006, IEEE, pp 9–pp
Hearty P, Fenton N, Marquez D, Neil M (2009) Predicting project velocity in XP using a learning dynamic bayesian network model. IEEE Trans Softw Eng 35(1):124–137
Hussain I, Kosseim L, Ormandjieva O (2013) Approximation of cosmic functional size to support early effort estimation in agile. Data and Knowledge Engineering 85:2–14
Idri A, Hosni M, Abran A (2016) Systematic literature review of ensemble effort estimation. J Syst Softw 118:151–175
Jahedpari F (2016) Artificial prediction markets for online prediction of continuous variables. PhD thesis, University of Bath, Bath
James G, Witten D, Hastie T, Tibshirani R (2015) An Introduction to Statistical Learning with Applications in R. Springer Texts in Statistics. Springer, New York
Jonsson L, Borg M, Broman D, Sandahl K, Eldh S, Runeson P (2016) Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. Empir Softw Eng 21(4):1533–1578
Jørgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Trans Softw Eng 33(1):33–53
Karner G (1993) Resource estimation for objectory projects. Objective Systems SF AB, p 17
Kocaguneli E, Menzies T, Keung JW (2012) On the value of ensemble effort estimation. IEEE Trans Softw Eng 38(6):1403–1416
Kultur Y, Turhanm B, Bener AB (2008) ENNA: software effort estimation using ensemble of neural networks with associative memory. In: 16th ACM SIGSOFT
Lee D (2016) Alternatives to p value: confidence interval and effect size. Korean Journal of Anesthesiology 69(6):555–562
Li Y, Yue T, Ali S, Zhang L (2017) Zen-ReqOptimizer: A search-based approach for requirements assignment optimization. Empir Softw Eng 22(1):175–234
Logue K, McDaid K, Greer D (2007) Allowing for task uncertainties and dependencies in agile release planning. In: 4th Proceedings of the Software Measurement European Forum, pp 275–284
Lokan C, Mendes E (2014) Investigating the use of duration-based moving windows to improve software effort prediction: A replicated study. Inf Softw Technol 56(9):1063–1075
MacDonell SG, Shepperd M (2010) Data accumulation and software effort prediction. In: ACM-IEEE International Symposium on Empirical Software Engineering and Measurement

Magazinius A, Börjesson S, Feldt R (2012) Investigating intentional distortions in software cost estimation–an exploratory study. J Syst Softw 85(8):1770–1781
Mahnic V, Hovelja T (2012) On using planning poker for estimating user stories. The Journal of Systems and Software 85:2086–2095
Minku L, Yao X (2013) Ensembles and locality: Insight on improving software effort estimation. Inf Softw Technol 55(8):1512–1528
Miyazaki Y, Takanou A, Nozaki H, Nakagawa N, Okada K (1991) Method to estimate parameter values in software prediction models. Inf Softw Technol 33:239–243
Neill J (2008) Why use effect sizes instead of significance testing in program evaluation? http://www.wilderdom.com/research/effectsizes.html, accessed: 2018-07
Nunes N, Constantine L, Kazman R (2011) iUCP: Estimating interactive-software project size with enhanced use-case points. IEEE Software 28(04):64–73
Palmer S, Felsing J (2002) A Practical Guide to Feature-driven Development. Prentice Hall, Upper Sadle River
Papatheocharous E, Papadopoulos H, Andreou A (2010) Feature subset selection for software cost modelling and estimation. Eng Intell Syst 18:233–246
Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in python. J Mach Learn Res 12:2825–2830
Pendharkar P, Subramanian G, Rodger J (2005) A probabilistic model for predicting software development effort. IEEE Trans Softw Eng 31(7):615–624
Perols J, Chari K, Agrawal M (2009) Information market-based decision fusion. Manag Sci 55(5):827–842
Pikkarainen M, Haikara J, Salo O, Abrahamsson P, Still J (2008) The impact of agile practices on communication in software development. Empir Softw Eng 13(3):303–337
Santana C, Leoneo F, Vasconcelos A, Gusmão C (2011) Using function points in agile projects. In: International Conference on Agile Software Development. Springer, pp 176–191
Schwaber K, Sutherland J (2016) The scrum guide (2013). Dostopno na:. http://www.scrumguidesorg/docs/scrumguide/v1/scrum-guide-uspdf (dostop 28–4–2016)
Shmueli G, Bruce P, Patel N (2016) Data Mining for Business Analytics: Concepts, Techniques, and Applications with XLMiner. Wiley, Hoboken
Stapleton J (1997) Dynamic systems development method. Addison-Wesley, Boston
Usman M, Mendes E, Weidt F, Britto R (2014) Effort estimation in agile software development: a systematic literature review. In: 10th International Conference on Predictive Models in Software Engineering, pp 82–91
Usman M, Mendes E, Börstler J (2015) Effort estimation in agile software development: a survey on the state of the practice. In: 19th International Conference on Evaluation and Assessment in Software Engineering
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(2):101–132
VersionOne (2016) 10th annual state of agile report. Technical report
Vidgen R, Wang X (2009) Coevolving systems and the organization of agile software development. Inf Syst Res 20(3):355–376
Wen J, Li S, Lin Z, Hu Y, Huang C (2012) Systematic literature review of machine learning based software development effort estimation models. Inf Softw Technol 54(1):41–59
Wolpert DH (1992) Stacked generalization. Neural networks 5(2):241–259

Onkar Malgonde is an Assistant Professor at the Operations Management & Information Systems, College of Business, Northern Illinois University. He received the B.Engg degree in Information Technology from the University of Pune, and the MS and PhD in Information Systems from the University of South Florida. His research interests include software engineering, machine learning, and digital platforms. His research has been published in Workshop on Information Systems and Technology (WITS), International Conference on Design Science Research in Information Systems and Technology (DESRIST), and Americas Conference of Information Systems (AMCIS).

Kaushal Chari is a professor and the associate dean of research and professional programs for the USF Muma College of Business. Previously he served as the chair of the Information Systems and Decision Sciences Department from 2006-2013. Chari's research program covers three broad areas: software engineering, business intelligence and distributed systems. Chari's work has been published in a variety of academic journals, including Empirical Software Engineering, IEEE Transactions on Software Engineering, Management Science, Information Systems Research, and the INFORMS Journal on Computing. Chari served as the associate editor of MIS for Interfaces journal from 2002-2010, and as the vice chair of the INFORMS Information Systems Society from 2007-2009. He is the co-winner of the 2009 Design Science Award from INFORMS Information Systems Society. Chari obtained a B.Tech in mechanical engineering from the Indian Institute of Technology Kanpur, followed by an MBA and PhD from the University of Iowa.