Cost-Effective Supervised Learning Models for Software Effort Estimation in Agile Environments

Kayhan Moharreri, Alhad Vinayak Sapre, Jayashree Ramanathan, Rajiv Ramnath
Department of Computer Science and Engineering
The Ohio State University, Columbus, OH
{moharrer, sapre, jayram, ramnath}@cse.ohio-state.edu

***Abstract*—Software development effort estimation is the process of predicting the most realistic effort required to develop or maintain software. It is important to develop estimation models and appropriate techniques to avoid losses caused by poor estimation. However, no method exists that is the most appropriate one for Agile Development where frequent iterations involve the customer causing time consuming estimation process. To address this an automated estimation methodology called "Auto-Estimate" is proposed complementing Agile's manual Planning Poker. The Auto-Estimate leverages features extracted from Agile story cards, and their actual effort time. The approach is justified by evaluating alternative machine learning algorithms for effort prediction. It is shown that selected machine learning methods perform better than Planning Poker estimates in the later stages of a project. This estimation approach is evaluated for accuracy, applicability and value, and the results are presented within a real-world setting.**

***Index Terms*—Software Estimation; Planning Poker; Agile Software Estimation and Planning; Machine Learning; Classifier**

I. Introduction

Software development effort estimation is the process of predicting the most realistic effort required to develop or maintain software based on incomplete, uncertain and/or noisy input. Within Agile Development projects with a high degree of customer involvement at every iteration, it is particularly important to have accurate estimates. It provides status visibility for the stakeholders, comparing planned progress with the actual progress. Effort estimates are used as input to project plans, iteration plans, budgets, and bidding rounds. Typical effort estimation is classified into three major categories: (a) Formal and Algorithmic, (b) Expert, and (c) Analogue estimations. Formal and Algorithmic models (ex. COCOMO, Function Point analysis) rely on the experience gained in previous software projects. They connect size and effort values by means of explicit function forms, by applying regression analysis method. Most widely used are linear and exponential dependence [9]. Expert estimation methods (ex. Wideband Delphi[22], Planning Poker[14]) are based on consultation with one or more people considered as experts in software development. These estimates are produced based on judgments. Analogue estimation methods are based on analogies between the new project and some already completed ones. Comparisons are made between the suggested project and similar projects based on known cost, time and effort data. Analogue models (ex. ESTOR, ANGEL) require as much data as possible concerning implemented projects [9]. Accurate estimates also help avoid schedule-stress-related quality problems. About 40% of all software errors have been found to be caused by human stress; those errors could have been avoided by scheduling appropriately and by placing less stress on the developers [10]. Failure to correctly interpret the meaning of an initial mismatch between project goals and project estimates is one of the most common wasted opportunities in software development. This pressures the project team to eventually commit to shorter schedules, even though this situation could be avoided by accurate effort estimation at the beginning[19].

The goal of this research is to develop an automated estimation method for Agile story cards which effectively applies existing machine learning algorithms to historical estimation data collected from humans. This "Auto-Estimate" method enhances manual Planning Poker [6], the most commonly used estimation method in Agile environments. Auto-Estimate makes use of features extracted from story cards used in an Agile development environment. Our research contributions are as follows: (a) Increasing the estimation accuracy by reducing the impact due to poor estimates; (b) Demonstrating that Auto-Estimate enhances Planning Poker in the later iterations of the project; (c) Establishing the importance of writing well-structured story cards; and (d) Creating an estimation tool with easy data collection which does not require any data other than story cards and previous effort estimates. In section II the related work and the state of the art research in software effort estimation will be discussed; in section III the methodology for Auto-Estimate will be proposed, in section IV the key steps in building and validating Auto-Estimate are shown. Results and discussions on improved estimation methods will be presented in section V. Finally conclusions are in section VI.

II. Related Work

Predictive modeling has been widely used within IT enterprise environments, applications of which range from recommendation-based IT service delivery and support [23], [20] to defect prediction for commercial software [7]. This research in particular aims at software effort estimation which has been researched in different software design and development industries. Although there have been different estimation models developed thus far, no single model has been shown to be out performing others. As a result there is a need for new forecasting models due to dynamic changes in the environment. Recently more research attention was drawn to statistical modeling in order to address general effort estimation challenges. Authors in [8] provided a comparative analysis consisting of more than a dozen learning models against different benchmarks. Also in [15] an ensemble learning methodology was proposed for effort estimation to boost estimation accuracy. Within Agile environments, however, there has not been much research for effort estimation using supervised learning models.

Bakele and Turhan in [3] reported that well-structured and detailed use cases provide higher confidence and accuracy for effort prediction models. Also they concluded that software effort estimation must be handled using an evolving system rather than a static one. Motivated by these papers, we noted that in addition to the features extracted from story cards, project related features in Agile environment can be used to enrich the estimation process. Development platform (i.e. Operating system, and development tools), and development techniques (i.e., CMMI, ISO, etc) are features that have been shown to have most impact on the estimation process.

Agile tries to minimize the impact of insufficient estimation accuracy by ensuring that the most important functionality is developed first. This is achieved through a flexible development process with short iterations. In Agile a user story is the unit at which software features are estimated and developed. Each story is in the language of the customer, and typically written on an index card. The cards serve as reminders for conversations to be had about the features. The details are then fleshed out later in the conversations, and conveyed and documented in the form of tests [2].

Planning Poker is an important method for effort estimation in Agile. Participants include the programmers, testers, analysts on the team. At the start of Planning Poker, each estimator is given a deck of cards. Each card has written on it one of the valid estimates. Each estimator may, for example, be given a deck of cards with numbers from the Fibonacci Series 0, 1, 2, 3, 5, 8, 13 and 21. These non-linear sequence gaps reflect the greater uncertainty associated with estimates for larger units of work [6]. For each user story to be estimated, a moderator who is the product owner reads the description and then answers any questions that the estimators have. Each estimator privately selects cards representing his or her estimate. Cards are not shown until each estimator has made a selection. At that time, all cards are simultaneously turned over and shown so that all participants can see each estimate. If estimates differ, the high and low estimators manual their estimates. The objective here is to learn what they were thinking about. The goal is for the estimators to converge on a single estimate that can be used for the story [6].

According to a comprehensive survey [2], the motivations for using Planning Poker can be any of the following: (a) more information is uncovered in the discussion; (b) individual estimation avoids first-estimate bias; (c) reflects the team's ability to complete the tasks, and (d) the whole team feels ownership of the estimates.

III. Research Methodology

A. Problem Statement

Based on our analysis of different aspects of Planning Poker, we conclude that the following areas need to be addressed rigorously. **Magnitude of Relative Error** (**MRE**) is a widely used measure for evaluating the estimation accuracy of different models. For a **single estimate**, it is defined as the following equation:

$$MRE = \frac{|\text{ Actual Effort} - \text{Estimated Effort }|}{\text{Actual Effort}} \tag{1}$$

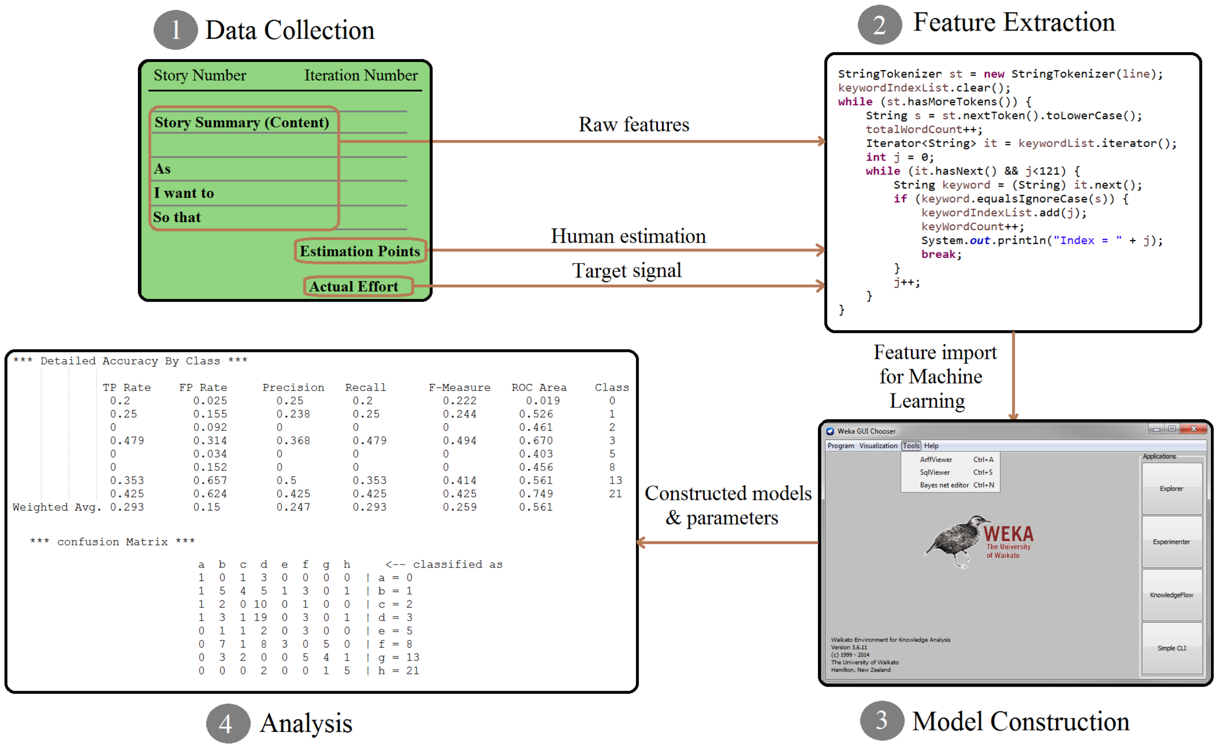
Also **Mean Magnitude of Relative Error** (**MMRE**) is used to quantify the accuracy achieved **overall** with the improved method. In this study, based on the estimation data collected from the enterprise for Planning Poker, the MMRE value was 106.81% which is considerably large and the goal is to reduce it. Also according to our data, it was observed that in Planning Poker over-estimates are the most common cases among all; precisely with the following distribution: Accurate:24%, Overestimate:40%, Underestimate: 36%. Details are provided in section V.

A drawback of Planning Poker identified was that even though it takes every developer's estimate into consideration, the bias towards estimates from experts could not be avoided. In other words, in the absence of an expert in the team, the accuracy of estimates decreases substantially. Furthermore, Planning Poker is less accurate when there is no previous experience from similar tasks. (a) If the developers suggest similar estimates for a story where they have no prior experience, they might feel a false sense of security that the estimate is reasonable. Or, (b) the developers become more cautious, taking the various possibilities mentioned by all developers into account, and therefore end up overestimating the complexity. Another shortcoming of Planning Poker is the group polarization effect. After participating in a discussion, members tend to advocate more extreme positions than individuals who did not participate in any such discussion. This offers a possible explanation as to why Planning Poker increases extremes.

B. Research Contributions

The current working model, i.e. Planning Poker involves individual estimation based on the story card followed by multiple rounds of discussion before an estimate is fixed and the developers can then start the development of the features described in the story card. Based on the conducted employee surveys [21], it has been concluded that Planning Poker works better when there are more experts in the development team. However, experts are not always easy to find in abundance while forming a team, and therefore the need for more number of experts becomes a problem. The goal with Auto-Estimate is to eventually fully automate the estimation, thus reducing the need of estimates from experts.

In case of Agile, more and more story card data becomes available with the progress of the project. Auto-Estimate's learning algorithms train on this data from story cards and the



```
StringTokenizer st = new StringTokenizer(line);
keywordIndexList.clear();
while (st.hasMoreTokens()) {
    String s = st.nextToken().toLowerCase();
    totalWordCount++;
    Iterator<String> it = keywordList.iterator();
    int j = 0;
    while (it.hasNext() && j<121) {
        String keyword = (String) it.next();
        if (keyword.equalsIgnoreCase(s)) {
            keywordIndexList.add(j);
            keyWordCount++;
            System.out.println("Index = " + j);
            break;
        }
        j++;
    }
}
```

```
*** Detailed Accuracy By Class ***

                TP Rate   FP Rate   Precision   Recall    F-Measure   ROC Area   Class
                 0.2       0.025     0.25        0.2       0.222       0.019      0
                 0.25      0.155     0.238       0.25      0.244       0.526      1
                 0         0.092     0           0         0           0.461      2
                 0.479     0.314     0.368       0.479     0.494       0.670      3
                 0         0.034     0           0         0           0.403      5
                 0         0.152     0           0         0           0.456      8
                 0.353     0.657     0.5         0.353     0.414       0.561      13
                 0.425     0.624     0.425       0.425     0.425       0.749      21
Weighted Avg.    0.293     0.15      0.247       0.293     0.259       0.561

  *** confusion Matrix ***

                                a  b  c  d  e  f  g  h    <-- classified as
                                1  0  1  3  0  0  0  0  | a = 0
                                1  5  4  5  1  3  0  1  | b = 1
                                1  2  0 10  0  1  0  0  | c = 2
                                1  3  1 19  0  3  0  1  | d = 3
                                0  1  1  2  0  3  0  0  | e = 5
                                0  7  1  8  3  0  5  0  | f = 8
                                0  3  2  0  0  5  4  1  | g = 13
                                0  0  0  2  0  0  1  5  | h = 21
```

Fig. 1. Methodology for developing Auto-Estimate

actual effort data. By the later stages of the project there will be larger sample of story cards and human estimates available where the model statistically becomes more equipped to infer the true relationship between the features and the actual effort signal. The need for human estimates can practically be eliminated at that point, possibly with robust accuracy for estimations.

IV. Methodology for Developing Auto-Estimate

As shown in Figure 1 the method for selecting the best learning techniques for Auto-Estimate is as follows. (1) Data Collection: using story cards, human estimates, and actual effort data (2) Feature Extraction: extracting key *tokens* using text analysis (3) Model Construction: using extracted features and human estimates along with actual effort to train classifiers, and (4) Analysis: measuring performance to select the best classifiers.

A. Data Collection

The required story card features, human estimation data, and actual effort data were accessed through iteration logs stored for each team using IBM Rational Team Concert. The data was collected from 10 different teams including the following features on each of the story cards: (a) Story card ID, (b) Project, (c) Planned story points (human estimation), and (d) Story card summary (text). To enable *supervised learning* the actual effort is considered as the *target signal*.

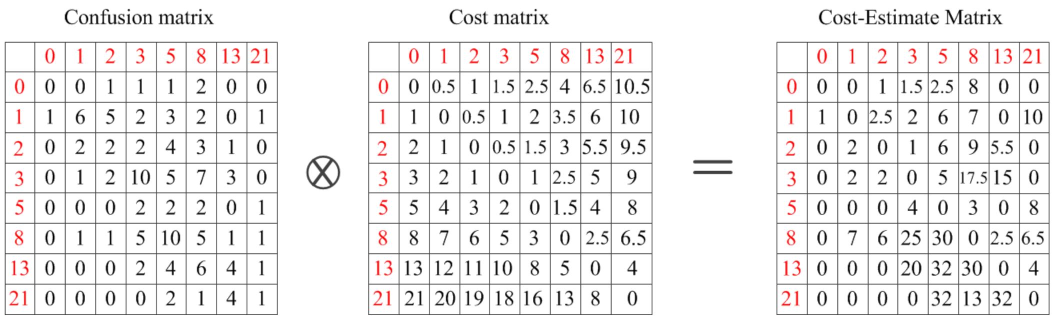
B. Feature Extraction

*1) **Token Extraction**:* According to a previous experiment [1], presence of word tokens in the story card was found out to be important features for effort estimation. Tokens represent a crucial part of the story content. Unigram language modeling with respect to tokens was used as a key component of our estimation model. Unique words and their frequencies were extracted from the set of one hundred and twenty three story cards. After applying an upper and lower threshold of frequency, 752 tokens were chosen for further analysis. The thresholds were applied to remove extremes with respect to word frequency.

*2) **Features Chosen**:* After the extraction of tokens, the following story card features were chosen for further analysis:
▽ *TF-IDF of each token*: “Term Frequency – Inverse Document Frequency” of a word is a metric to quantify significance of a word which takes into account word frequency and the inverse of the counts of documents containing that word.
▽ *Estimated points from Planning Poker*: This feature is selected because we want to use Planning Poker estimates as our supporting feature. Although we know that these estimates are not accurate, a combination of weak features is often used to make a classifier more accurate.
▽ *Number of tokens in the story*: More tokens implies more important functionality to be addressed.
▽ *Story card priority (enumerated as integer)*: Priority (from the customer’s perspective) for the feature to be developed may influence the effort to be put in.

C. Model Construction using supervised learning algorithms

The features extracted are used as inputs to Weka [12] for several classification algorithms. The process of how we selected the learning algorithms is described below:



Confusion matrix

	0	1	2	3	5	8	13	21
0	0	0	1	1	1	2	0	0
1	1	6	5	2	3	2	0	1
2	0	2	2	2	4	3	1	0
3	0	1	2	10	5	7	3	0
5	0	0	0	2	2	2	0	1
8	0	1	1	5	10	5	1	1
13	0	0	0	2	4	6	4	1
21	0	0	0	0	2	1	4	1

$\otimes$

Cost matrix

	0	1	2	3	5	8	13	21
0	0	0.5	1	1.5	2.5	4	6.5	10.5
1	1	0	0.5	1	2	3.5	6	10
2	2	1	0	0.5	1.5	3	5.5	9.5
3	3	2	1	0	1	2.5	5	9
5	5	4	3	2	0	1.5	4	8
8	8	7	6	5	3	0	2.5	6.5
13	13	12	11	10	8	5	0	4
21	21	20	19	18	16	13	8	0

$=$

Cost-Estimate Matrix

	0	1	2	3	5	8	13	21
0	0	0	1	1.5	2.5	8	0	0
1	1	0	2.5	2	6	7	0	10
2	0	2	0	1	6	9	5.5	0
3	0	2	2	0	5	17.5	15	0
5	0	0	0	4	0	3	0	8
8	0	7	6	25	30	0	2.5	6.5
13	0	0	0	20	32	30	0	4
21	0	0	0	0	32	13	32	0

Fig. 2. Example of Cost-estimate matrix (as the Hadamard product of the Confusion matrix and the Cost matrix)

▽ **Naïve Bayes** - A Naïve Bayes classifier is a probabilistic classifier based on Bayes' theorem with an assumption of independent features given the class. Despite its simplicity, Naïve Bayes battles the high dimensionality of the data with the conditional independence assumption which can often outperform more sophisticated classification methods. [5].

▽ **J48 Decision tree** - J48 is an open source Java implementation of the C4.5 algorithm in Weka. It is an algorithm used to generate a decision tree. Decision trees are fitting well to the training data when the tanning set is not linearly separable.

▽ **Random Forest** - Random Forest is a group classifiers that consists of many decision trees and outputs the class that is the statistical mode of the classes output by individual trees. According to [4], Random Forest is suited for learning from large datasets and avoids overfitting better than decision trees.

▽ **Logistic Model Tree** - A logistic model tree (LMT) is an algorithm for supervised learning tasks, which combines linear logistic regression and tree induction. In some previous experiments [17], it was exhibited that LMT produces classification models that are more accurate than those produced by C4.5, Classification and Regression Trees (CART) and simple logistic regression on real world datasets.

Also there were two types of feature sets used by these algorithms. In "semi-independent prediction" the human estimations based on Planning Poker are included as an input feature where the purpose is to see if access to these estimates can improve the quality of predictions. In "independent prediction" input features do not include the Planning Poker estimates where the purpose is to come up with independent predictions purely based on story cards. We have used the following statistical measures to verify the accuracy of our classifiers over our test data: Precision, Recall, F-Measure, and Confusion Matrix [16].

D. Cost-estimate analysis

Our approach for analysis was to find supervised learning models that could outperform manual Planning Poker in terms of the number of correct estimates. However, none of the methods were able to substantially outperform Planning Poker. We realized that the key deficiency in our early analysis was that the performance of the alternative methods (listed above) was based only on the 'hit or miss' numbers. That means methods were evaluated only based on the number of correct estimates and unduly penalized even if they were almost correct. To address this, we leveraged the notion of the confusion matrix and the cost matrix which are explained below, thus achieving a more rigorous model evaluation.

*1) **Confusion Matrix**:* The confusion matrix is part of the output of each classifier. Each column of the matrix represents the number of instances in a predicted class, while each row represents the number instances in an actual class.

*2) **Cost Matrix**:* The cost matrix was devised based on the penalties due to underestimation and overestimation. Notable studies [18] and [11] asserted that the penalty for underestimation is at least twice as much as that for overestimation. Taking this fact into consideration, the cost matrix is constructed in a similar way as the confusion matrix with the following key differences: The matrix contains penalty values for misclassification where the penalty for correct classification is 0. The penalty for classification as an underestimate is the magnitude of the difference between the actual value and the predicted value. Similarly, the penalty for classification as an overestimate is half of the magnitude of the difference.

*3) **Cost-Estimate Matrix**:* The cost-estimate matrix was derived by multiplying each element of confusion matrix with its corresponding element in the cost matrix (known as the *Hadamard product* [13]). This is illustrated in Figure 2. Also the value in each cell of this matrix is considered as the cost incurred due to that particular estimation inaccuracy. Thus, the sum of all values in the matrix yields the aggregate cost incurred by a particular classifier. This aggregate cost is further used for comparison among different methods.

V. Results and Analysis

In this section, the results of our experiments are stated and analyzed from different perspectives such as aggregate cost, mean magnitude of relative error, and distribution of estimates. Furthermore, our views are presented about whether Planning Poker should be eliminated entirely or it should be used in conjunction with Auto-Estimate.

A. Comparison of approaches - Aggregate Cost

After applying different classification methods to the training data, each method generated a confusion matrix, which was used for the cost-estimate analysis. The result of Cost-estimate analysis was the aggregate units of cost incurred by using that classification algorithm. Figure 3 shows the total costs incurred by different classification methods sorted in

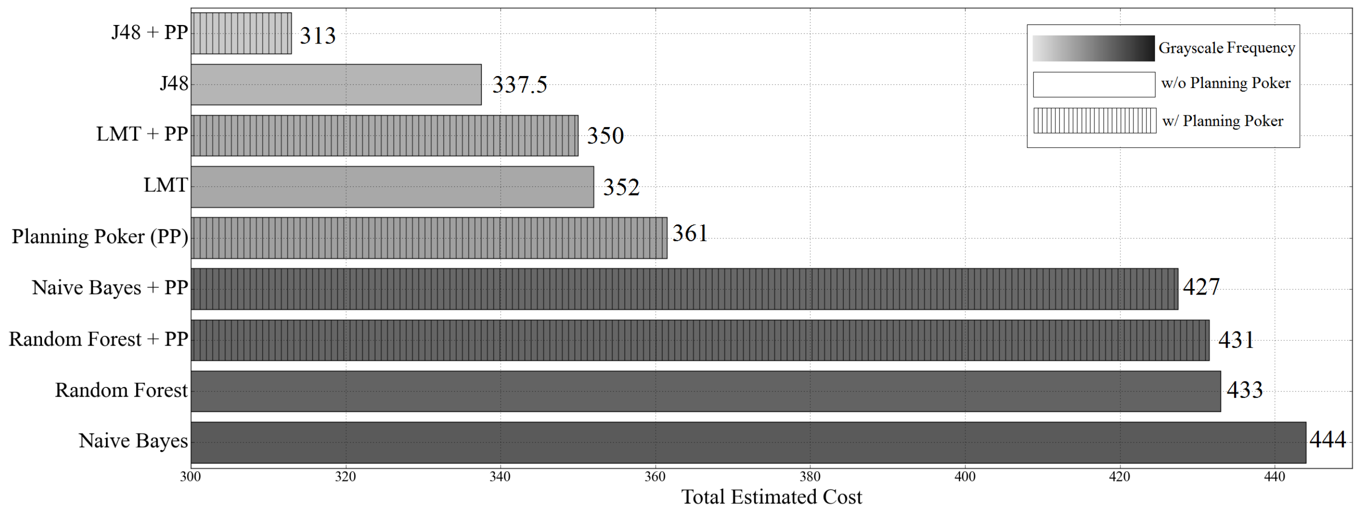


Fig. 3. Aggregate costs incurred by different classification methods (with and without Planning Poker – i.e. PP)

ascending order. Based on these results, the following methods were more cost-effective as compared to manual Planning Poker: (1) J48 + PP (i.e. J48 Decision Tree with Planning Poker estimates); (2) J48 (i.e. J48 Decision Tree without Planning Poker estimates); (3) LMT + PP (i.e. Logistic Model Tree with Planning Poker estimates); (4) LMT (i.e. Logistic Model Tree without Planning Poker estimates).

Figure 3 also reveals the fact that *including human estimates as a feature does consistently result in more cost-effective estimations*. Moreover, our observation of J48 Decision tree is that the algorithm was able to well correlate the tokens with actual effort. Secondly, the cost-estimate analysis proved that although some of the classification methods were outperformed by manual Planning Poker, two of the methods were able to cut down the costs. Thus showing promise in reducing the losses incurred due to poor estimation. In contrast, Random Forest performed poorly suffering from small training set.

B. Comparison of approaches - Error

A standard measure of estimation accuracy is Mean Magnitude of Relative Error (MMRE) (Equation 1). Table I presents the MMRE values associated with each of the methods. From Table I, it can be deduced that the following methods have a significant lower MMRE than manual Planning Poker: (1) J48+PP (2) J48 (3) Random Forest (4) LMT+PP. In line with cost-estimate results, here in terms of MMRE, J48 (both with and without PP augmentation) outperforms manual Planning Poker. In addition, we observed that half of the methods in Table I were able to cut down the MMRE values, thus increasing the estimation accuracy. This proved the key part of our hypothesis that is *estimation accuracy improves with the help of supervised learning*.

C. Comparison of approaches – Distribution of estimates

The distribution of estimates was also analyzed in order to verify the suitability of selecting classification methods for Auto-Estimate. Figure 4 illustrates a bubble chart representing probability distribution of estimates' outcome resulted from different classification methods. These distributions (as opposed to the hit or miss distributions) are highly effective when rigorous error analysis is needed. Each column represents a probability distribution of the estimate outcome corresponding to a particular classification method. As per Figure 4, J48+PP outperforms other methods since (1) it has the largest probability of correct estimates, and (2) it has a greater probability for its over-estimates as compared to its under-estimates. As mentioned earlier, over-estimates are preferred over under-estimates since slight over-estimation is not detrimental to an Agile project. Also per Figure 4, any method on the right of manual Planning Poker has shown to have a higher under-estimation probability than manual Planning Poker itself, and they also have shown lower correct-estimation probabilities than the manual Planning Poker; both of which are undesirable for Agile projects. In contrast, desirable methods on the left of manual Planning Poker such as J48+PP, LMT, and LMT+PP have lower under-estimation probabilities while maintaining higher correct estimation probabilities.

D. Eliminate planning poker or use it as an enhancement?

According to all previous comparisons, we come to the conclusion that *the following three classification methods are consistently superior to manual planning poker*: (1) J48 +PP, (2) J48, and (3) LMT+PP. Thus they should be included in a classification ensemble for Auto-Estimate methodology. An inherent characteristic of J48 and LMT that came to benefit here is that decision trees generally perform well on small training sets.

Inclusion of human estimates in the input helped most of the methods to perform well as compared to their counterparts. We generalize this interpretation as the following trade-off: Including manual estimates improves the performance of automated software effort estimation at the expense of running

TABLE I
MMRE FOR DIFFERENT CLASSIFICATION METHODS

Classification Method	MMRE
J48 + Planning Poker	**91.75**%
J48	92.32%
Random Forest	97.33%
LMT + Planning Poker	103.28%
Manual Planning Poker [Baseline]	106.81%
Random Forest + Planning Poker	108.95%
LMT	125.26%
Naïve Bayes + Planning Poker	180.09%
Naïve Bayes	204.40%

labor-intense Planning Poker to produce manual estimations out of story cards. On the flip side, eliminating Planning Poker from automated effort estimation costs the project with less accurate estimates, but at the same time benefits the project with complete automation of the estimation process saving substantial human time and labor.

VI. Conclusion

This research proposes 'Auto-Estimate' based on a comparison of alternate methods to enhance Agile's manual Planning Poker for effort estimation. We have shown that the J48+PP, J48, and the LMT+PP all performed better than manual Planning Poker. There were essential points leveraged: (1) Including human estimates as a feature improves performance. (2) Consideration of the error distribution and the aggregate cost is shown to be significant when comparing alternative classification methods. (3) As more data becomes available with each iteration, Auto-Estimate gets better trained on the story cards, human estimates, and actual effort data. Therefore by the later stages of the project the algorithm is more reliable than manual Planning Poker estimates and thus suitable as a tool for augmenting human effort estimation. For future work we propose research with larger datasets, and using features which were not used in this experiment (developers' demographics, story criticality, and other system and framework aspects).

VII. Acknowledgment

This research is supported by the NSF under Grant No. 0753710, the CERCS I/UCRC and industry sponsors.

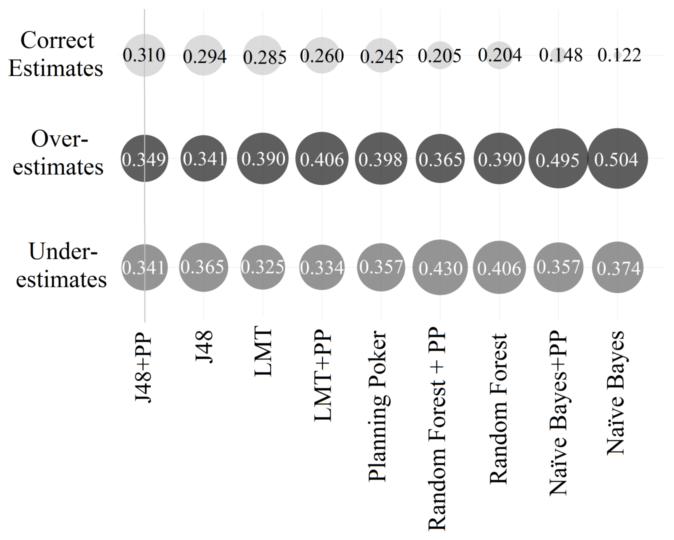


Fig. 4. Outcome distributions of estimates for different classifiers

References

[1] Pekka Abrahamsson, Ilenia Fronza, Raimund Moser, Jelena Vlasenko, and Witold Pedrycz. Predicting development effort from user stories. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, pages 400–403. IEEE, 2011.

[2] Nils C Augen. An empirical study of using planning poker for user story estimation. In *Agile Conference, 2006*, pages 9–pp. IEEE, 2006.

[3] Bilge Başkeleş, Burak Turhan, and Ayşe Bener. Software effort estimation using machine learning methods. In *Computer and information sciences, ISCIS 2007*, pages 1–6. IEEE, 2007.

[4] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *25th ICML*, pages 96–103. ACM, 2008.

[5] Rich Caruana and Alex Niculescu. Empirical comparison of supervised learning algorithms. In *23rd ICML*, pages 161–168. ACM, 2006.

[6] Mike Cohn. *Agile estimating and planning*. Pearson Education, 2005.

[7] Marco D'Ambros and Michele Lanza. An extensive comparison of bug prediction approaches. In *Mining Software Repositories (MSR), 7th IEEE Working Conference on*, pages 31–41. IEEE, 2010.

[8] Karel Dejaeger, Wouter Verbeke, David Martens, and Bart Baesens. Data mining techniques for software effort estimation: a comparative study. *Software Engineering, IEEE Transactions on*, 38(2):375–397, 2012.

[9] Gavin R Finnie, Gerhard E Wittig, and Jean-Marc Desharnais. A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models. *Journal of Systems and Software*, 39(3):281–289, 1997.

[10] Robert L Glass. Is field: Stress up, satisfaction down. *Software Practitioner*, pages 1–5, 1994.

[11] Eliyahu M Goldratt. *Critical chain:[a business novel]*. North River Press Great Barrington, MA, 1997.

[12] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[13] Roger A Horn. The hadamard product. In *Proc. Symp. Appl. Math*, volume 40, pages 87–169, 1990.

[14] Martin Höst and Claes Wohlin. An experimental study of individual subjective effort estimation and combinations of the estimates. In *Proceedings of the 20th international conference on Software engineering*, pages 332–339. IEEE Computer Society, 1998.

[15] Ekrem Kocaguneli, Tim Menzies, and Jacky W Keung. On the value of ensemble effort estimation. *Software Engineering, IEEE Transactions on*, 38(6):1403–1416, 2012.

[16] Ron Kohavi and Foster Provost. Glossary of terms. *Machine Learning*, 30(2-3):271–274, 1998.

[17] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. *Machine Learning*, 59(1-2):161–205, 2005.

[18] Patricia K Lawlis, Robert M Flowe, and James B Thordahl. A correlational study of the cmm and software development performance. *Crosstalk*, 8(9):21–25, 1995.

[19] Steve McConnell. *Software estimation: demystifying the black art*. Microsoft press, 2006.

[20] Kayhan Moharreri, Jayashree Ramanathan, and Rajiv Ramnath. Probabilistic sequence modeling for trustworthy it servicing by collective expert networks. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, volume 2. IEEE, 2016.

[21] Alhad Vinayak Sapre. Feasibility of automated estimation of software development effort in agile environments. Master's thesis, The Ohio State University, 2012.

[22] Andrew Stellman and Jennifer Greene. *Applied software project management*. "O'Reilly Media, Inc.", 2005.

[23] Liang Tang, Tao Li, Larisa Shwartz, Florian Pinel, and Genady Ya Grabarnik. An integrated framework for optimizing automatic monitoring systems in large it infrastructures. In *Proceedings of the 19th ACM SIGKDD*, pages 1249–1257. ACM, 2013.