International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-3, September 2019

Feature Engineering for Enhanced Model Performance in Software Effort Estimation

Sreekumar P. Pillai, Radharamanan T., Madhukumar S.D.

Abstract*: Many new methodologies have been defined in the last two decades in the domain of Software Effort Estimation. They include manual methods based on expert judgment, analogy-based models, parametric models, regression models, machine learning models, and more recently, deep learning models. Except for manual methods, all other models depend heavily on data. Lack of quality data in this domain is a motivation to explore means to optimize the sparse data available. Machine learning algorithms depend on domain features, and their ability to represent and model the domain, to solve the problems irrespective of whether it is classification or regression, image, or voice synthesis. There is continued research for the best representation of the issue through the right feature space. While most of the traditional research rely on the original dataset and concentrate more on feature selection, modern-day approaches explore creating additional features that have the potential to extend the models representational space.This research builds on our last research exploring the potential to improve Software Effort Estimation accuracy by employing engineered features in addition to the original ones. The features are created manually based on the literature. Through the engineered features, we captured additional representational features such as missingness and proportion of categorical data available in the dataset. We present the rationale for the features generated and compare the prediction accuracy between a model using the original dataset and the engineered data set.Our experiments in Feature Engineering is innovative in the Software Estimation domain and the results conclusive establishing its use in predicting Software Effort. We report an improved accuracy of 38% with engineered features at PRED(15), and 11% improvement at PRED(20). The quantitative growth that we have been able to achieve in terms of accuracy is promising enough for this to be adopted as a standard in future research on the subject and practical applications.

Keywords*: Software Effort Estimation, Software Cost Estimation, Effort Prediction, Feature Engineering, Generalized Linear Model, Artificial Neural Network, Random Forests.

I. INTRODUCTION

The 2014 Standish Chaos report claims only 16.2% of the IT projects are successful globally. Even among projects executed by large companies, the success rate is only 9%. And, yet when these projects are completed, Estimating Software Effort accurately has always been a challenge to project managers and research scholars. Lack of proper estimation capability has been identified as one among the top 5 reasons for project failures [1].

In the last two decades, many estimation techniques and models have been discussed in Software Effort Estimation. Jorgensen and Shepperd [2] reveal 11 estimation methods [1]. Regression-based models, Expert Judgment, Analogy, and Machine Learning models add to the models already defined in the industry. With the availability of data and the democratization of Machine Learning in the last few years, algorithmic models have found a place in the software estimation space. Even as early as 2013, 79% of primary studies on estimation used data-oriented approaches using Machine Learning [3].

Parametric estimation approaches build a model around a feature space that represents the different facets of the problem domain that helps predict the target variable. The advantage of Machine learning techniques is their ability to model relationships between the dependent variable and the independent variables, which in many cases are very complex and diverse. In the case of software effort estimation, this involves understanding the effect of the critical variables (Table I) and their relationship on the software development effort.

The more recent approaches in Machine Learning include Artificial Neural Network, regression trees, and gradient boosting tress. Ensemble models are latest in the group [4] where multiple models are combined to improve the predictive accuracy of the resultant model for cumulative efficiency.

Despite the existence of all these methodologies and models, challenges exist in Software Effort Estimation (SEE) due to insufficient accuracy, inconsistent metrics being used to evaluate the models, and lack of enough project management data. While there is a need to standardize [5]the accuracy metrics, the major challenge faced by researchers and industry experts alike is the lack of quality data related to project execution [6], and scholars are forced to depend on globally available datasets to build their prediction models. Independent organizations have attempted to solve this challenge by collating datasets for analysis. The leading ones in this context are ISBSG, Finnish, Tukutuku, and the Desharnais datasets. In this regard, it is relevant that we point our research in the direction of exploring how feature engineering can help data leakage and improve prediction within the limitations of the available data.

Manuscript published on 30 September 2019
* Correspondence Author
Sreekumar P. Pillai*, Research Scholar, School of Management Studies, National Institute of Technology Calicut, Kozhikode, Kerala, India. Email: sreekumar.pillai@hotmail.com
Dr. T. Radharamanan, Head of Department, School of Management Studies, NITC, Kozhikode, Kerala, India.
Dr. S.D. Madhukumar, Professor, Dean (SW), Dept. of Computer Science, NITC, Kozhikode, Kerala, India.

II. BACKGROUND WORK

Along with Machine Learning, there is a continuing interest in the notion of feature engineering, which involves creating new features from existing ones to improve the performance of an algorithmic model. In one of the early books on the subject published in 1998, [7] "Feature Extraction, Construction and Selection, there is a whole section dedicated to Feature Construction laying the foundation for new means of achieving the domain representation.

The definition of Feature Construction is laid out in one of the seminal papers in Feature Engineering by Matheus et al.[8] as: "Creating a framework based on four critical aspects: detection, selection, generalization, and evaluation." Their CITRE framework performed feature construction using decision trees and real domain knowledge as constructive biases.

In all domains, and parallel to the growth of Machine Learning, scholars have started to focus on Feature engineering to improve upon the existing data, as distinct from feature selection, well known in the ML science.

A. Feature Selection Vs. Feature Engineering

Feature engineering constructs suitable features from the existing ones leading to the increased predictive performance of the ML model and transforms the current data using arithmetic and aggregate operators. Transformation of the data helps scale a specific feature or convert a non-linear relation between a variable and a target class into a linear association, to make it easier to be consumed by the ML algorithm. Kang.et.al [9] in their book on Feature-Oriented Domain Analysis" states that features should capture the user's understanding of capabilities of the application domain. Brazdil, Pavel discusses many induction-based techniques in his paper "Constructive Induction of Continuous Spaces" [10].

Turner.et.al[11] coined the term 'Feature Engineering" in their paper on the topic in 1999, and they set out "promotion of features as First-class objects within the software process" as the goal for feature engineering.

Liu.et.al summarizes many discretization techniques to achieve improved accuracy as a feature engineering technique for continuous values [12].

Christopher.et.al [13] elaborated feature Engineering techniques in knowledge-based construction, and this was further employed in a "question-answer" selection solution by Aliased Severyn [14] evidencing a model performance improvement of 22%.

As distinct from Feature Engineering, Feature selection is a way to identify variable subsets that reproduce the observed values of a dependent variable, i.e., those subsets that are, for a proposed problem, the most useful to obtain a more accurate regression model [15]. When done manually, this requires a large amount of domain knowledge to understand the interaction and occurrence of the features within the domain space can be remodeled for the additional representational power to make an impact on the model performance.

Feature engineering can be automated as done in unsupervised machine learning models. In this instance, the ML engine creates hundreds of features that may not be intuitive to the human brain, as in the case of image recognition models.

B. Feature Engineering Techniques

A few of the widely used feature engineering techniques include [16]:

1) *Missing value imputation*
2) *Imputing Methods*
   Mean or Median Imputation, Random Sample Imputation, End Tail Imputation, Add Nan Binary Imputation, Categorical variable imputation, Frequent Category Imputation
3) *Variable Transformation methods*
   Log Transformer, Reciprocal Transformer, Exponential Transformer, Box-Cox Transformer
4) *Encoding Methods*
   Count Frequency, Ordinal Encoding, One Hot Encoding, Rare Label Encoding
5) *Discretization Methods*
   Discretization, Equal Frequency Discretizer, Equal Width Discretizer, Decision Tree

The standard procedures performed on the data as part of pre-processing is not considered feature engineering since it is a part of the default Machine Learning pipeline. In the case of software effort estimation, the input space is not large enough to warrant employing an automated feature selection mechanism.

C. Software Size and Function Points

Allan Albrecht defined Function Points (FP) in 1979 in Measuring Application Development Productivity at IBM. Function Points is one of the oldest and still the most pertinent unit employed to measure the size of software; also quantifies the business functionality of a software application. In Function Point Analysis, the user interaction with the system is broken down to its purest form as processes that performs a finite task. These processes are then categorized into any one of five types, such as External Output, External Query, External Input, Internal Files, and External Interfaces. These processes are measured for their complexity based on the quantity of information managed and the number of references it makes. The points are assigned to each of these processes based on the complexity. The total size of an application is the sum of Function Points represented by each of these processes. Function Points are relevant since they quantify an intangible product such as a piece of software. Function Point also has the distinction of being predictive (this can be estimated before the software is created), as against SLOC (Software Lines of Code) that is measurable from the source code of an application that is existing. Industry and scholars accept Function Point as accurate since its fault tolerance is less than 10% [17], [18].

D. The ISBSG Dataset

The data that we used in this study is from ISBSG (Release 1, February 2018). Of late, this dataset has been used in many studies, ISBSG aims to provide IT industry data to better software processes and products, as stated on their website.

This dataset contains information about 8261 projects executed globally and collected from volunteering organizations. The projects were submitted from 26 countries [19].

Guevara et al. [3] conducted a systematic mapping study on ISBSG dataset and listed the 20 most frequently used variables in Software Effort Estimation studies. Table I lists a minimum subset we selected from this mapping study along with the newly engineered features from the current experiment.

E. Ensembles and Stacked Ensembles

Machine Learning ensembles are set of trained models trained to perform software effort estimation to improve the prediction accuracy [20]. One of the main reasons for clubbing multiple models is to make use of the performance capabilities of the different models. Combining the performance of the different models by averaging may or may not provide a good result but reduces the risk of poor performance. In certain other cases, ensemble learners help manage huge amount of data [21]. Ensembles can also help by employing the right kind of model for the data and help increase the overall prediction performance.

We use an ensemble of all the four other individual models discussed GLM, GBM, ANN and Random Forest to form an ensemble. We follow the averaging the outcomes of the individual models as followed by some of the scholars [22],[23].

We use a technique known as Stacked Generalization for ensemble learning. In the two layers adopted by this procedure, the first layer composes of base learners and the meta learners on the second layer uses layer one as input. The stacked ensemble uses cross-validation to generate and n x L matrix of k-fold values with n rows and L number of base learners (Vika, 2011). For this reason, the cross-validation schema should be identical across all the rows in an ensemble. The H2o.ai statistics platform implements stack ensembles in their DeepWater module (Phan et al., 2017). We use a stacked
ensemble model generated from the two datasets the compare the performance of the engineered dataset.

III. Problem Statement

Making optimum use of available data has become an acute necessity with the advent of machine learning models, especially critical with regards to Software Effort Estimation since there is a dearth for quality data, and it becomes pertinent to maximize the representation power of data through the right Feature Management techniques. Our experiment is intended to fill this gap and identify additional features that have the best representational power to maximize estimation accuracy.

IV. Methodology

Our experiment methodology is illustrated in Figure I. We selected quality records from the ISBSG R1 2018 dataset based on their quality parameters. After the cleaning procedure, we chose the minimum variable subset that can represent the software estimation domain, giving us 10 distinct variables. From this list, normalized product delivery rate (norm.pdr), and project year (proj.year) variables were used only to derive the previous year's productivity. We made a copy of this dataset to engineer new columns as elaborated further in the "Feature Engineering Experiment" section marked it as the Engineered Dataset. Both these datasets were passed separately through the outlier elimination and normalization procedures as part of the feature selection process. We removed collinear columns from the engineered dataset during a Principal Component Analysis (PCA). In Table I, the "used" column denotes the final set of variables that were used for the comparison experiment. As the next step, we constructed two ensemble models from the original and the engineered dataset using the H2o.ai platform. The identical metrics gathered from the same models facilitated reporting comparative metrics as provided in Table IV.

The figure, graph, chart can be written as per given below schedule.

A. Tools

We used the statistical platform H2o.ai to conduct all statistical experiments. H2O provides Deep Water, a framework for GPU-accelerated deep learning. H2O Deep Water leverages prominent open source deep learning frameworks, such as MXNet, TensorFlow, and Case, as backends. As part of the H2O platform, Deep Water can take advantage of grid-search, model checkpointing, and ensembles. H2O can process billions of data rows in-memory, even with a small cluster. H2O's platform includes interfaces for R, Python, and Java among others with common machine learning algorithms, such as generalized linear modeling (GLM) (linear regression, logistic regression among others), Naive Bayes, principal components analysis, k-means clustering, and word2vec. All data preprocessing and experiment were conducted using R and R-Studio environments.

B. Feature Selection

The ISBSG 2018 R1 version of the ISBSG dataset that we used for this study holds 105 features relating to software projects. Previous studies have evidenced that a minimal subset of the feature population can represent the variability from the perspective of Effort Estimation. [3], [24].A brief description of the 8 critical variables identified and their relevance to the experiment is detailed in the following sub-sections.

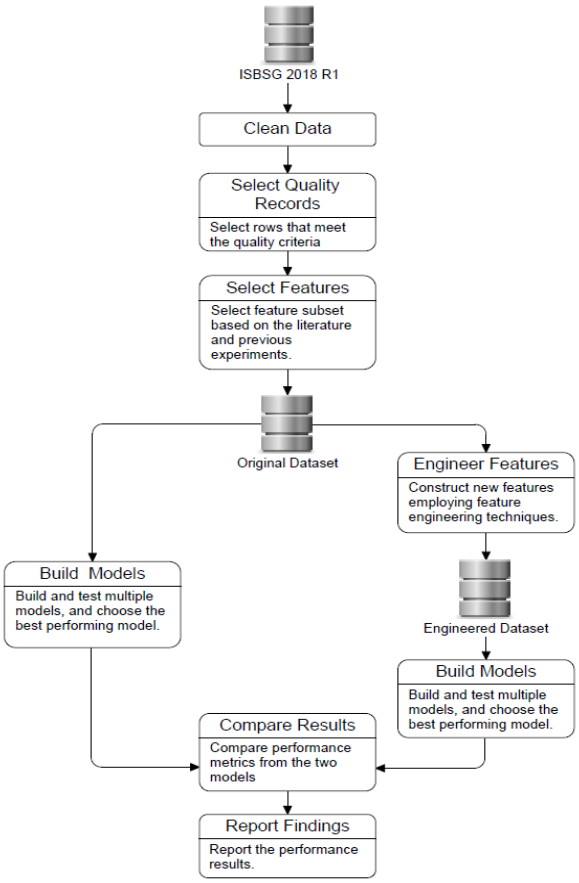

Figure I:Engineering Methodology

1) Un-Adjusted Functional Size

The un-adjusted functional size refers to the size of the software developed and has the most impact on development effort and is exponentially correlated to the project effort [25]. We have used unadjusted size variable from ISBSG. The adjusted size re-evaluates the size to account for the complexity of the environment in which the software is developed.

2) Value Adjustment Factor

The VAF is a multiplication factor in Function Point Analysis that helps scale the size variable to accommodate for the varying complexities involved in the development of a software product. VAF can vary from 0.65 to 1.35. It uses 14 General System Characteristics (GSC) to derive a measure of the complexity of a project. The original size is multiplied with VAF to get the adjusted Functional Size. C.J.Lokan has studied the impacts of VAF [17], concluding that the relationship of VAF with function points should be reviewed. We retain this variable since it represents the nature of the environment and impacts the overall effort.

3) Development Type

The development type variable categorizes the projects based on the nature of the activity. The development of a new project is significantly different in terms of complexity from a re-development or enhancement work. Software productivity varies based on the type of development work [26].

4) Language Type

The programming language employed in projects is categorized based on their generations. New generation languages offer high productivity and require only fewer lines of code. There are five labels in the lang.type variable in the dataset.

5) Project Elapsed Time

The project duration in calendar months impacts the effort required to develop software. The relation between software effort and schedule is non-linear as concluded by empirical studies [27]. The effort required to develop the software and the calendar duration is two distinct features.

6) Team Size Group

The team.size.gp variable represents the average number of members employed in the project team during development. The size of the team has a significant impact on the project effort. While major projects cannot be executed with a small group, a larger team brings communication overhead and other dysfunctions and have been the subject for many studies [28]. Jiang and Comstock [29] evidenced that that team size and the language type are significant determinants in software effort estimation.

7) Industry Sector

The industry sector is a grouping variable that classifies the ISBSG projects across 18 distinct domain areas. The complexity of software products/features needed by industry domains are distinct and reflect its characteristics. Software productivity also varies across industry domains, [30]. Many scholars have proposed stratification or layering of projects and building individual models to enhance the accuracy of estimation models [31],[27].

8) Work Summary Effort

The summ.weffort variable represents the effort required to produce software of defined size and is the dependent variable. Since above 90% of the cost of software development is dependent on human effort, estimating this becomes the primary focus, the mathematical problem in any software effort estimation study.

C. Data Pre-processing

The data preprocessing was done identically for both the original and engineered datasets. In both cases, we used only those rows that had passed the quality standards based on the rating set by ISBSG.

Table 1:Original and Engineered Features and Operations

No.	Variable	Description	Type	Transform	Re-Group	Orig./ Engd.	Used
1	lang.type	Language Type	Factor	None	Yes	Orig.	Yes
2	dev.type	Development Type	Factor	None	Yes	Orig.	Yes
3	ind.sector	Industrial Sector	Factor	None	Yes	Orig.	Yes
4	proj.year	Project Year	Num, factor	None	No	Orig.	No
5	na.is	NA in IS	Num, factor	None	No	Engd.	No
6	na.dt	NA in DT	Num, factor	None	No	Engd.	No
7	na.lt	NA in LT	Num, factor	None	No	Engd.	Yes
8	na.vaf	NA in VAF	Num, factor	None	No	Engd.	No
9	na.fs	NA in FS	Num, factor	None	No	Engd.	No
10	na.pet	NA in PET	Num	Yes	No	Engd.	No
11	cnt.is	No. of labels in IS	Num	Yes	No	Engd.	No
12	cnt.dt	No. of labels in DT	Num	Yes	No	Engd.	No
13	cnt.lt	No. of labels LT	Num	Yes	No	Engd.	No
14	team.size.gp	Average Team Size	Num, factor	None	No	Orig.	Yes
15	value.af	Value Adjustment Factor	Num	Yes	No	Orig.	Yes
16	funct.size	Functional Size	Num	Yes	No	Orig.	Yes
17	summ.weffort	Work Effort	Num	Yes	No	Orig.	Yes
18	pre.avg.pdr	Productivity of the previous year	Num	Yes	No	Engd.	Yes

1) Data Quality Rating

ISBSG uses two attributes ratings on, Data Quality, and Unadjusted Function Points to denote the reliability of the sizing measure. Projects are classified from A to D (from where the submitter satisfy all criteria specified by ISBSG, to where the data has severe shortcomings). ISBSG recommends only projects classified as 'A' or 'B' to be considered for statistical analysis. In our analysis, all data rows rated as 'C' or 'D' were removed.

2) Unadjusted Function Point Rating

Only projects classified as A or B are considered for this research as recommended by ISBSG and other scholars [3].

3) Count Approach

The counting approach is the technique used to measure the size of a project. All variants of the Function Point are retained. The 'Recording Method' affirms that the organization has collected the information first hand or derived them from other parameters. We used only where the information has been confirmed as directly observed and recorded in the dataset.

Table II: Descriptive statistics of the engineered dataset

No	Variables	Mean	Sd	Median	Mad	Min	Max	Range	Skew	kurtosis	se
1	value.af	0.01	0.06	0	0	-0.44	0.26	0.70	0.37	9.52	0
2	funct.size	4.08	0.98	4.12	0.84	1.06	6.74	5.67	0.55	0.46	0.02
3	prev.avg.pdr	2.52	0.94	2.46	0.75	-0.71	5.19	5.89	0.58	0.59	0.02
4	summ.weffort	5.86	0.84	5.93	0.79	1.96	7.89	5.93	-0.51	0.55	0.02

n=3041, cnt.dt=Count of dev type, value.af = Value Adjustment Factor, proj.elap.tm = Project Elapsed Time, funct.size = Functional Size, prev.avg.pdr = Previous year average product delivery rate, summ.weffort = Summary work effort

4) Outliers

To eliminate outliers, we used a multivariate outlier detection procedure using Cook's distance [32], [23]. Outliers are detected based on influences from all numerical variables together. Yeong et al. [33] mention a procedure for outlier detection using multivariate analysis, further elaborated by Kannan and Manoj[34]. The influence of each data point (row) in the dataset is computed based on the equation:

$$D_i = \frac{\sum_{j=1}^{n}\left(\hat{Y}_j - \hat{Y}_{j(i)}\right)^2}{\text{p x MSE}} \quad (1)$$

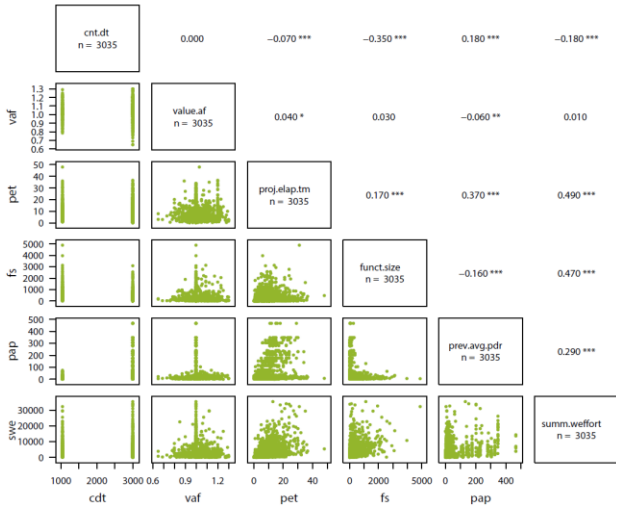

Figure II: Correlational Plot

n=3041, cdt=Count of dev type, vaf = Value Adjustment Factor, pet = Project Elapsed Time, fs = Functional Size, pap = Previous year average product delivery rate, summ.weffort = Summary work effort

Points with cook's distance greater than four times the mean were considered outliers and removed from the engineered dataset [35].

5) Normalization

We normalized both the original as well as the engineered dataset using Box-Cox transformation function, which gave the best outcome compared with the other options we tried employing log-normal and cube-root transformations.

6) Modeling and Testing

We used the train and test sets to build and test our linear regression model by splitting the dataset using the 80:20 ratio. We built the model using the training set reserving the test set to validate the model accuracy with independent data.

7) Model Accuracy

The accuracy of the models is evaluated and reported using the following nomenclature from previous studies on the subject. For any single project,

- Magnitude of Relative Error (MRE) Measures the absolute estimation accuracy and is defined as [27]:

$$MRE = \frac{|ActualEffort - PredictedEffort|}{ActualEffort} \quad (2)$$

- The Mean Magnitude of Relative Error (MMRE) is expressed as:

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|ActualEffort - PredictedEffort|}{ActualEffort} \quad (3)$$

- Prediction (PRED) is determined for n projects; the prediction at level p is defined as:

$$PRED(p) = \frac{k}{n} \quad (4)$$

k is the number of projects where the MRE is less than or equals p. Desirable MMRE and PRED thresholds were first published by [36], and subsequently by other scholars [37].

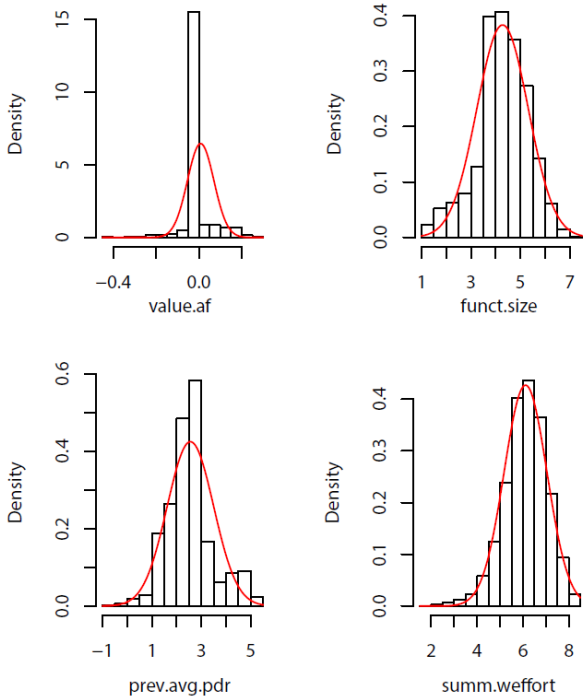

Figure III: Histogram of numerical variables after Box-cox transformation

V. THE FEATURE ENGINEERING EXPERIMENT

We performed the following feature engineering techniques on the original data as part of the experiment.

A. Missing Value Imputation

Standard imputation has been performed on ISBSG dataset by Sentas and Angelis [26], Shepperd and Cartwright [38],[39], Song et al. [40], Sehra et al. [41], and recently by Mittas et al.[22]. We performed imputation of missing values using Buuren's, `Multivariate Imputation' method [42] as implemented in the mice' R' package.

Within the ISBSG dataset version we used, specific projects have captured the unadjusted Function Points, and others measured Adjusted FP along with VAF. We removed rows only where both these values are non-existent. In cases where two of these values are present, we have derived the missing values based on equation (1). This helps us retain close to 30% of the rows missing VAF in the original dataset.

Data imputation is done for Language Type (lang.type) and Team Size Group(team.size.gp) variables based on multivariate imputation further evidenced in the papers by Cartwright[39], and recently by Lee [43].

The feature engineering experiment was conducted in the following order to arrive at the optimum set of features.

B. New Transformed Features

Software Product Delivery Rate (PDR) is a metric used by ISBSG to denote the productivity of software Development based on each product. This is a derived measure from the quantity of software produced and the time taken to produce it. This is a lag measure since this is available only once the project is complete. We hypothesize that the historical productivity of a team is indicative of the future effort required by any team.

features also means that there should be adequate records matching all the combinations of these features. Since this is not the case with the ISBSG dataset, we accept this loss of information bringing down the number of records to 3041.

C. Missing Value Imputation

We wanted to capture "missingness" of the features to ascertain their representational value to the prediction capability of our model. We created an is.Na column for the categorical variables in the dataset.

We imputed the original ind.sector feature with values

Table III: Regrouped Labels in Categorical Variables

No.	Feature	Labels	Abb.	New Label
1	Development Type	Enhancement	Enhan	Enhan
		New Development	NewD	Dev
		Re Development	ReDev	Dev
2	Industrial Sector	Banking	Bk	(Bk)(Ma)
		Communication	Co	(MH)(Co)
		Construction	Cn	(Cn)(PS)
		Defence	Df	(Ut)(Df)
		Education	Ed	(Mi)(Ed)(WR)(Fi)
		Electronics & Computers	EC	(SI)(EC)
		Financial	Fi	(Mi)(Ed)(WR)(Fi)
		Government	Go	(Ot)(Go)
		Insurance	In	(In)
		Logistics	Lg	(Lg)
		Manufacturing	Ma	(Bk)(Ma)
		Medical & Health Care	MH	(MH)(Co)
		Mining	Mi	(Mi)(Ed)(WR)(Fi)
		Others	Ot	(Ot)(Go)
		Professional Services	PS	(Cn)(PS)
		Service Industry	SI	(SI)(EC)
		Utilities	Ut	(Ut)(Df)
		Wholesale & Retail	WR	(Mi)(Ed)(WR)(Fi)
3	Language Type	2GL	2GL	(2GL)
		3GL	3GL	(3GL)
		4GL	4GL	(4GL)(ApG)
		ApG	ApG	(4GL)(ApG)

The software productivity rate is also dependent on the Industry sector (complexity of software varies based on the nature of the domain), the programming language generation, and the skill availability in the team. To account for these variations in the software production rate, we engineered a feature that combined all four elements. Our engineered productivity average referred to the mean of the productivity Figures for the preceding year grouped against all these dependent categorical features. Due to this transformation, we have lost information for the year 1989, the first year in the dataset since there is no previous year data to substitute. Taking the mean of software production rate against all these

from the same column using CART. The CART algorithm is based on Classification and Regression Trees by Breiman et al. (1984)[44]. A CART tree is a binary decision tree constructed through splitting each node of a tree, recursively starting with the base node. The 'R' package MICE implement CART [42] for imputation.

D. Imputation for Missing Value Adjustment Factor

In the original dataset, Fields are missing for this column. We imputed the missing values in this column using the formula:

$$AFP = VAF * UAFP \quad (5)$$

Where, AFP = Adjusted FP, UAFP = Unadjusted Function Point. Wherever this value could not be derived, we substituted the value with 1. This substitution implies that no external factors influencing the software development effort and the factor representing the general system characteristics equate to 1.

E. Stratification based on Homogeneous Labels

Stratification has improved the accuracy of effort estimates [45], and [46]. Additionally, fewer number of variables increases the model's manageability. We conducted the ANOVA test for categorical variables and found the presence of homogenous groups. We employed the multiple comparison procedure proposed by Donoghue [47] to visualize the homogeneous groups and further implemented in the `R' package MultCompView," by Graves et al.[48].

Table IV: Experiment Results

Category	Dataset/ Dataset	Original Dataset	Engineered Dataset	Improvement
Model Metrics	MSE	0.6458	0.2157	-0.4301
	RMSE	0.8036	0.4644	-0.3392
	MMRE	0.1091	0.0649	-0.0442
	r^2	0.7800	0.7900	0.0100
Prediction Accuracy	PRED(5)	0.1607	0.4877	0.3270
	PRED(10)	0.4098	0.8095	0.3997
	PRED(15)	0.6689	0.9245	0.2556
	PRED(20)	0.8738	0.9721	0.0983
	PRED(25)	0.9541	0.9852	0.0311
	PRED(30)	0.9885	0.9918	0.0033

MSE = Mean square error, RMSE = Root mean square error, MMRE = Mean magnitude of relative error, NB: Negative values reflect diminishing performance

1) Language Type - lang.type

The one-way ANOVA test displays a statistically significant difference between the different groups of the lang.type variable (F (3.4037) = 5.526, p = 0.000876). Post-Hoc testing using the Bonferroni adjusted P-value method suggested identical mean effort values for 2GL and ApG languages. We combined these two levels into a single value called ApG.

2) Industry Sector – ind.sector

One-way ANOVA test done on the ind.sector variable shows statistically significant difference between its groups (F (13.4027) = 19.42, p = 2e-16). The Post-Hoc tests using the Bonferroni adjusted P-value method suggested grouping homogeneous industry sectors to simplify the model and reduce errors. We grouped the categorical variables as in Table III based on the ANOVA results.

F. Removing Irrelevant Columns

Specific columns were found to be irrelevant after the Feature Engineering step, and we removed them from the Engineered dataset.

1) Development Type - dev.type

The one-way ANOVA test shows a statistically significant difference between the groups (F(2.4038) = 103.7, p = 2e-16). Post-Hoc testing using the Bonferroni adjusted P-value method suggested identical mean values for Re-Development and New Development projects. We combined these two levels into a single value called Development.

The team.size.gp and max.team.size are two variables that represent the size of the team employed in the software development process. During PCA, we dropped the max.team.size variable as it was redundant substantiated by its high correlation with the team.size.gp variable. We decided to retain the team.size.gp variable, that is intuitive and meaningful to researchers and practitioners alike. None of the other variables display collinearity.

We also dropped all variables that captured the missing information, and label counts except for Industry Sector (ind.sector) and Development Type (dev.type) that showed representational value.

The proj.year and norm.pdr variables were removed from the dataset since they were irrelevant after deriving the mean product delivery rate of the previous year (prev.avg.pdr). The variables that were employed for the final experiment are listed in Table I.

G. Exploratory Statistics of the Engineered Dataset

The descriptive statistics of the engineered dataset in Table II. We transformed the variables using the Box-Cox transformer implemented in the R package MASS, that has been able to rectify the skew in the data to an acceptable level.

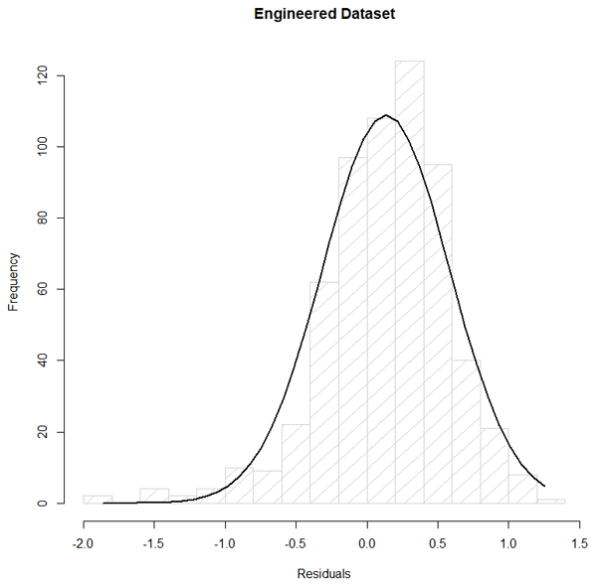

Figure IV: Residual Plot

H. Modeling and Testing

We created two different models, and the results from these models for both the original and engineered datasets is given in Table IV. From this, we derived the metric values in equations 1 to 3 for the original as well as the engineered dataset. The training R2 value shows 2.8% improvement and evidence that the engineered features can represent the domain more efficiently. The predicted values using the new model increases the accuracy of prediction by 20% with 5% tolerance PRED(5).

From the original dataset, we were only able to retain 795 rows after feature selection and the preprocessing steps. Through feature engineering, we have been able to keep 3041 with additional gains in accuracy.

The result from the original dataset is lower by 2% in prediction, especially in the lower threshold region. In the standard Prediction metrics, the engineered dataset shows a 5% increase in the accuracy metrics.

Our analysis of the residuals is presented in Fig.IV. The residuals (errors) from the prediction are very close to a normal distribution showing no signs of Heteroscedasticity [49].

VI. Results

Our results evidence that Software effort estimation can be improved through Feature Engineering and that the mean software production rate of the team from the preceding year has important representational ability to model the relationship of the independent variables and the development effort.

Table IV summarizes the results of our experiments. The key metrices from both datasets are shown in the table along with the improved performance. The R^2 and the prediction value at 15% tolerance has improved by 1% and 38% respectively. The performance improvement is high at the lower tolerance range and reflects a higher performance capability. The improvement diminishes from PRED(20) downwards, the standard tolerance against which scholars compare Software predictions [50],[27].

VII. Limitations

This research is based on the publicly available dataset and does not consider the environmental or organizational aspects of software development in terms of development skills or process maturity. While the model encapsulates the global nature of the current day software development process, individual models catering to the specific development organizations might evidence to produce more accurate estimates. ISBSG has limitations to provide a countrywide classification of productivity due to confidentiality agreements which pose an impediment to ascertain whether national productivity impacts the estimated effort, thereby the cost of software development, geographically.

VIII. Conclusion

The results of our experiments on the ISBSG dataset R1 are conclusive to employ feature engineering to build better performing estimation models in the future. The new features that we derived increase the representational value of the dataset by 2.8% and increase the estimation accuracy by 20% in the minimum threshold region. The research also evidences the use of past productivity in the estimation of future projects and substantiates the intuitive and logical hypothesis that historical productivity can be relied on to estimate the next development effort.

REFERENCES

1. The Standish Group, The Standish group: the chaos report, Project Smart (2014) 16. doi:10.1016/S0895-7061(01)01532-1. arXiv:arXiv:1011.1669v3.
2. M. Jorgensen, M. J. Shepperd, M. Jorgensen, M. J.Shepperd, M. Jorgensen, M. J. Shepperd, M. Jorgensen, M. J. Shepperd, A Systematic Review of Software Development Cost Estimation Studies, IEEE Transactions on Software Engineering 33 (2007) 33-53. doi:10.1109/TSE.2007.256943.arXiv:arXiv:1011.1669v3.
3. F. Gonzalez-Ladron-de Guevara, M. Fernandez-Diego,C. Lokan, The usage of ISBSG data fields in software effort estimation: A systematic mapping study, The Journal of Systems and Software 113 (2016) 188-215.doi:10.1016/j.jss.2015.11.040.
4. E. Kocaguneli, T. Menzies, E. Mendes, Transfer learning in effort estimation, Empirical Software Engineering 20 (2015) 813-843. doi:10.1007/s10664-014-9300-5.
5. M. Ochodek, J. Nawrocki, K. Kwarciak, simplifying effort estimation based on Use Case Points, Information and Software Technology 53 (2011) 200-213.doi:10.1016/j.infsof.2010.10.005.
6. R. Jeffery, M. Ruhe, I. Wieczorek, using public domain metrics to estimate software development effort, Proceedings - 7th International Software Metrics Symposium, 2001 (2001) 16-27.doi:10.1109/METRIC.2001.915512.
7. H. M. Huan Liu (Ed.), Feature Extraction, Construction and Selection, A Data Mining Perspective, i ed., Springer Science Business Media LLC, 1998.
8. C. J. Matheus, I. L. Group, Constructive Induction on Decision Trees, Machine Learning (1983) 645-650.
9. P. Brazdil, Constructive Induction on Continuous Spaces, Feature Extraction, Construction and Selection (2011). doi:10.1007/978-1-4615-5725-8.
10. Y. Ou, Y. Xu, C.-a. Engineering, Feature Oriented Domain Analysis (FODA), Technical Report May, Carnegie-Mellon University Software Engineering Institute, 1990.

11. C. Reid Turner, A. Fuggetta, L. Lavazza, A. L. Wolf, A conceptual basis for feature engineering, Journal of Systems and Software 49 (1999) 3{15.doi:10.1016/S0164-1212(99)00062-X.
12. M. H.Liu, F.Hussain, C.L.Tan, Discretization: An Enabling Technique, Data Mining and Knowledge Discovery 6 (2002) 393-423.
13. C. Re, A. A. Sadeghian, Z. Shan, J. Shin, F. Wang,S. Wu, C. Zhang, Feature Engineering for Knowledge Base Construction, ARXIV (2014) 1-14. URL:http://arxiv.org/abs/1407.6439. arXiv:1407.6439.
14. A. Severyn, A. Moschitti, Automatic Feature Engineering for Answer Selection and Extraction, Empirical Methods in Natural Language Processing 13 (2013) 458-467.
15. R. F. Teofilo, J. P. A. Martins, M. M. C. Ferreira, Sorting variables by using informative vectors as a strategy for feature selection in multivariate regression, Journal of Chemometrics 23 (2009) 32-48.doi:10.1002/cem.1192.
16. A. Zheng, A. Casari, Feature Engineering for Machine Learning and Data Analytics - Principles and Techniques for Data Scientists, i ed., Oreilly, 2018. URL:http://bit.ly/featureEngineering for ML.
17. C. Lokan, An empirical analysis of function point adjustment factors, Information and Software Technology 42 (2000) 649-659. doi:10.1016/S0950-5849(00)00108-7.
18. M. Molani, A. Ghaffari, A. Jafarian, A new approach to software project cost estimation using a hybrid model of radial basis function neural network and genetic algorithm, Indian Journal of Science 7 (2014) 838-843.
19. ISBSG, Demographics of Development & Enhancement Repository, Technical Report, ISBSG, 2017.
20. E. Kocaguneli, T. Menzies, J. W. Keung, On the value of ensemble effort estimation, IEEE Transactions on Software Engineering 38 (2012) 1403-1416.doi:10.1109/TSE.2011.111.
21. R. Polikar, Ensemble based systems in decision making, Circuits and Systems Magazine, IEEE 6 (2006) 21-45. doi:10.1109/MCAS.2006.1688199.arXiv:arXiv:1011.1669v3.
22. N. Mittas, E. Papatheocharous, L. Angelis, A. S. Andreou, integrating non-parametric models with linear components for producing software cost estimations, Journal of Systems and Software 99 (2015) 120-134.doi:10.1016/j.jss.2014.09.025.
23. X. Y. Leandro L. Minku, Ensembles and locality: Insight on improving software effort estimation, Information and Software Technology 55 (2013) 1512-1528.doi:10.1016/j.infsof.2012.09.012.
24. K. Petersen, Measuring and Predicting Software Productivity: A Systematic Map and Review, Information and Software Technology 53 (2011) 317-343.doi:10.1016/j.infsof.2010.12.001.
25. M. Jorgensen, B. Boehm, S. Rifkin, Software development effort estimation: Formal models or expert judgment? IEEE Software 26 (2009) 14-19.doi:10.1109/MS.2009.47.
26. P. Sentas, L. Angelis, Categorical missing data imputation for software cost estimation by multinomial logistic regression, Journal of Systems and Software 79 (2006)404-414. doi:10.1016/j.jss.2005.02.026.
27. W. Rosa, R. Madachy, B. Boehm, B. Clark, C. Jones, J. Mcgarry, J. Dean, Improved Method for Predicting Software Effort and Schedule, International Cost Estimating and Analysis Association (ICEAA) (2014).
28. F. P. Brooks, The Mythical Man Month, Prentice Hall,2001.
29. Z. Jiang, C. Comstock, The Factors Significant to Software Development Productivity, International Journal of Computer, Electrical, Automation, Control and Information Engineering 25 (2007) 160-164.
30. A. Trendowicz, J. Munch, Chapter 6 Factors Influencing Software Development Productivity-State-of-the-Art and Industrial Experiences, Advances in Computers 77 (2009) 185-241. doi:10.1016/S0065-2458(09)01206-6.
31. O. Jalali, T. Menzies, D. Baker, J. Hihn, Column pruning beats stratification in effort estimation, Proceedings - ICSE 2007 Workshops: Third International Workshop on Predictor Models in Software Engineering, PROMISE'07 (2007). doi:10.1109/PROMISE.2007.3.
32. R. D. Cook, Detection of Influental Observation in Linear Regression, Technometrics 19 (1977) 15-18.
33. R. J. Yeong-Seok Seo, Doo-Hwan Bae, AREION: Software effort estimation based on multiple regressions with adaptive recursive data partitioning, Information and Software Technology 55 (2013) 1710-1725.doi:10.1016/j.infsof.2013.03.007.
34. K. S. Kannan, K. Manoj, Outlier detection in multivariate data, Applied Mathematical Sciences 9 (2015) 2317-2324. doi:10.12988/ams.2015.53213.
35. L. Lavazza, G. Valetto, Requirements-based estimation of change costs, Empirical Software Engineering 5 (2000) 229-243. doi:10.1023/A:1026590615963.
36. H. S.D. Conte, V.Y.Shen, Software Engineering Metrics and Models, benjamin/c ed., Benjamin/Cummings, Menlo Park, 1986, 1986.
37. D. Port, M. Korte, Comparative Studies of the Model Evaluation Criterions Mmre and Pred in Software Cost Estimation Research, Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (2008) 51-60.doi:10.1145/1414004.1414015.
38. M. Shepperd, M. Cartwright, Predicting with sparse data, IEEE Transactions on Software Engineering 27 (2001) 987-998. doi:10.1109/32.965339.
39. M. Cartwright, Data Imputation Techniques for Software Engineering : Case for Support, IEEE Transactions on Software Engineering (2003) 1-8.
40. Q. Song, M. J. Shepperd, X. Chen, J. Liu, can k-NN Imputation Improve the Performance of C4 . 5 With Small Software Project Data Sets ? A Comparative Evaluation, Journal of Systems and Software 81 (2008) 1-31. doi:10.1016/j.jss.2008.05.008.
41. S. K. Sehra, J. Kaur, Y. S. Brar, N. Kaur, Analysis of Data Mining Techniques for Software Effort Estimation, 2014 11th International Conference on Information Technology: New Generations (2014) 633-638.doi:10.1109/ITNG.2014.116.
42. S. Van Buuren, K. Groothuis-Oudshoorn, S. Van Buuren, K. Groothuis-Oudshoorn, mice :Multivariate Imputation by Chained Equations in R, Journal of Statistical Software VV (2011). URL: http://www.jstatsoft.org/.doi:10.18637/jss.v045.i03. arXiv:arXiv:1501.0228.
43. J. B. Taeho Lee, Taewan G, MND-SCEMP: An empirical study of a software cost estimation modeling process in the defense domain, Empirical Software Engineering 19 (2014) 213-240. doi:10.1007/s10664-012-9220-1.
44. L. Breiman, J. Friedman, C. Stone, R. Olshen, Classification and Regression Trees, The Wadsworth and Brooks-Cole statistics probability series, Taylor & Francis, 1984. URL: https://books.google.co.uk/books?id=JwQx-WOmSyQC.
45. V. Nguyen, B. Boehm, P. Danphitsanuphan, A controlled experiment in assessing and estimating software maintenance tasks, Information and Software Technology 53 (2011) 682-691. doi:10.1016/j.infsof.2010.11.003.
46. M. Tsunoda, S. Amasaki, A. Monden, Handling categorical variables in effort estimation, Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '12 (2012)99. doi:10.1145/2372251.2372267.
47. J. R. Donoghue, Implementing Shaffer's multiple comparison procedure for a large number of groups, Wiley Online 1998 (1998) i-38.
48. S. Graves, H.-P. Piepho, L. Sundar, D.-R. Maintainer, L. Selzer, Package 'multcompView' Visualizations of Paired Comparisons, R package http://CRAN.R-project.org/package=multcompView (2015).
49. D. Rodriguez, J. J. Cuadrado-Gallego, J. S. Aguilar Ruiz, using genetic algorithms to generate estimation models, 2006.
50. P. Jodpimai, P. Sophatsathit, C. Lursinsap, Estimating Software E ort with Minimum Features Using Neural Functional Approximation, Computational Science and Its Applications (ICCSA), 2010 International Conference on (2010) 266-273. doi:10.1109/ICCSA.2010.63.

AUTHORS PROFILE

Sreekumar P. Pillai is a research scholar (Ph.D.) in Software Effort Estimation employing Machine Learning techniques, holding a M.Phil. in Management and a Master's degree in Business Administration. He has over 20 years of industry experience in Information Systems development and software engineering. He is also a Data Scientist and solutionist. His areas of interests include Data Engineering, Data Science, Artificial Intelligence, Robotic Process Automation, and Software Engineering models in productivity, estimation, quality and risk. He has published papers and articles on machine learning and management and is very active in the technology community.

Dr.T.Radha Ramanan is Head of School of Management Studies, NITC, Kozhikode, Kerala, India. He holds a Ph.D. in Industrial Engineering from NIT Trichy and an M.Tech in IE from Regional Engineering College, Trichy. His areas of interest are Operations Management, Data Analysis, Decision Models, Performance measurement among others. He has produced 5 research scholars and supervising 10 more candidates. He has also guided more than two dozen PG projects. He has published around 15 papers in international journals of repute and more than 50 papers in various international conferences. He is also a reviewer of many international journals.

Dr.S.D.Madhu Kumar is Professor at Dept. of Computer Science and Dean (SW) at NIT Calicut. He has a Ph.D.in CSE from IIT Bombay and M.E. in Computer Science and Engineering from IISc, Bangalore. He has mentored 7 research scholars and currently supervising 5 candidates and has over 70 papers to his credit in international journals and conferences. He is a Member of the "Board of Studies, Computer Science", Cochin University of Science & Technology (CUSAT). His areas of interests include Distributed Computing, Cloud Computing, Big Data and Cloud Database and Software Engineering.