THE REPUBLIC OF TURKEY

BAHÇEŞEHİR UNIVERSITY

SOFTWARE DEVELOPMENT EFFORT ESTIMATION USING ENSEMBLE MACHINE LEARNING

Master's Thesis

OMAR HIDMI

ISTANBUL, 2016

THE REPUBLIC OF TURKEY

BAHÇEŞEHİR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

INFORMATION TECHNOLOGIES

SOFTWARE DEVELOPMENT EFFORT ESTIMATION USING ENSEMBLE MACHINE LEARNING

Master's Thesis

OMAR HIDMI

Supervisor: ASST. PROF. BETÜL ERDOĞDU ŞAKAR

ISTANBUL, 2016

THE REPUBLIC OF TURKEY

BAHÇEŞEHİR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
INFORMATION TECHNOLOGIES

Name of the thesis: SOFTWARE DEVELOPMENT EFFORT ESTIMATION USING ENSEMBLE MACHINE LEARNING

Name/Last Name of the Student: OMAR HIDMI

Date of the Defense of Thesis: May 25th, 2016

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Assoc. Prof. Nafiz ARICA
Graduate School Director
Signature

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Prof. Adem KARAHOCA
Program Coordinator
Signature

This is to certify that we have read this thesis and we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Committee Members	Signature
Asst. Prof. Betül ERDOĞDU ŞAKAR	_____
Asst. Prof. Görkem SERBES	_____
Asst. Prof. Tarkan AYDIN	_____

ACKNOWLEDGEMENT

It is a pleasure for me to thank people who have made the finish of this thesis possible; first and foremost, I would like to thank my wife Jinan for being so patient and helpful and show me endless love, my daughter Tuqa, my parents Hisham and Hanin and all my family in Palestine who encouraged me all the time. Without them, I never would have been able to reach this stage.

I would like to express my deepest gratitude both personally and professionally, to my supervisor Asst. Prof. Betül ERDOĞDU ŞAKAR, without her guidance and encouragement the completion of this study would be impossible.

I would also like to express a very special thanks to Meshale International Student Association for financially supporting me and giving me the chance to come here to Turkey and complete my higher studies.

Finally, I would like to thank all my friends for their motivation and support.

ISTANBUL, 2016 Omar Hidmi

ABSTRACT

SOFTWARE DEVELOPMENT EFFORT ESTIMATION USING ENSEMBLE MACHINE LEARNING

Omar Hidmi

Information Technology

Thesis Supervisor: Asst. Prof. Betül ERDOĞDU ŞAKAR

May 2016, 40 pages

In software engineering, the main aim is to develop a high quality projects that fall within scheduled time and budget, this procedure is called effort estimation. There are many techniques to estimate the effort including expert judgment and the use of machine learning techniques. Effort estimation is crucial and important for a company to do because hiring more people than needed will lead to loss of income, and hiring less people than needed will lead to delay of project delivery. The aim of this study is to estimate software effort objectively by using machine learning techniques instead of subjectively and time consuming estimation methods like expert judgment and estimation by analogy. The proposed solution mentioned in this study tries to overcome the problems of over-estimation and under-estimation by improving the software effort estimation process. This study will propose a model that uses two machine learning techniques which are Support Vector Machine (SVM) and K-Nearest Neighbor (*k*-NN) and combining them together using a boosting technique called AdaBoost for better effort estimation in new software development projects. Machine learning techniques have been applied to two publically available dataset which are Desharnais and Maxwell. Results show that SVM technique outperform k-NN technique, results also show much improvement in estimations when using AdaBoost.

Keywords: Machine Learning, Effort Estimation, K-Nearest Neighbor, Support Vector Machine, AdaBoost.

ÖZET

TOPLULUK YAPAY ÖĞRENME İLE YAZILIM GELİŞTİRME MAALİYET TAHMİNİ

Omar Hidmi

Bilgi Teknologi

Tez Danışman: Yrd. Doç. Dr. Betül ERDOĞDU ŞAKAR

Mayıs 2016, 40 sayfa

Yazılım mühendisliğindeki ana amaç belli bir zaman dilimi ve bütçe ile yüksek kaliteli projeler üretmektir, bu yönteme efor tahmini denir. Efor tahmini için uzman tahmini ve yapay öğrenme tekniklerinin kullanımı gibi oldukça farklı yöntemler bulunmaktadır. Efor tahmini firmalar için oldukça önemli ve kritik bir karardır. Çünkü fazla personel ile çalışılması durumunda firma finansal zarara uğrarken az personel ile çalışıldığında projenin uzaması ya da ertelenmesi gibi durumlar oluşabilir.

Bu projenin amacı uzman tahmini ya da benzerlikle tahmin gibi öznel ve zaman alıcı yöntemler yerine yapay öğrenme tekniklerini kullanarak yazılım efor tahminini nesnel hale getirmektir. Bu çalışmada önerilen çözüm ile yazılım efor tahmini sürecinde aşırı öğrenme ve az öğrenme sorunlarının üstesinden gelinmeye çalışılmıştır.

Bu çalışmada iki farklı yapay öğrenme tekniği, Destek Vektör Makinaları ve K-En Yakın Komşu algoritmaları hem ayrı ayrı denenmiş, hem de AdaBoost arttırma yöntemi bu iki algoritmanın sonuçlarını birleştirilerek yeni yazılım geliştirme projelerinin eforr tahminini yapacak modeller önerilmiştir. Yapay öğrenme teknikleri tüm kullanıcılara açık olan Desharnais ve Maxwell veri kümeleri üzerinde denenmiştir.

Anahtar Kelimeler: Yapay Öğrenme, Efor Tahmini, K-En Yakın Komşu, Destek Vektör Makinesi, AdaBoost.

CONTENTS

TABLES

FIGURES

ABBREVIATIONS

AdaBoost	:	Adaptive Boosting
ANN	:	Artificial Neural Networks
CART	:	Classification And Regression Trees
CBR	:	Case Based Regression
COCOMO	:	Constructive Cost Model
FP	:	Function Points
GA	:	Genetic Algorithm
KLOC	:	Kilo Lines Of Codes
K-NN	:	K – Nearest Neighbor
LSR	:	Least Square Regression
M5P	:	M5-Pruned
MART	:	Multiple Additive Regression Trees
MATLAB	:	Matrix Laboratory
PROMISE	:	The PRedictOr Models In Software Engineering data repository
RBF	:	Radial Basis Function
RI	:	Rule Induction
SVM	:	Support Vector Machine
SVR	:	Support Vector Regression
WEKA	:	Waikato Environment for Knowledge Analysis

1. INTRODUCTION

1.1 MOTIVATION

Estimation is really important in our lives; we make estimations approximately all the time, for example, kids unconsciously estimate how much time to finish their homework, parents estimate how much time to finish cooking and how much time to go to work. Of course time estimated always oscillates according to certain conditions and external factors, such as emergency situations, weather conditions, and so on. Also when we plan to open a new business, we estimate cost, time, and staff needed to finish a particular job. Same thing is in software development estimations.

Software project management is one of the most important parts of software engineering. One cannot guarantee that good software management would lead to project success. On the other hand, it is for sure that a project will fail with bad project management. Project failure can be dependent on different factors like late delivery, less or more estimated costs, fall short of customer requirements and last but not least wrongly estimated number of staff.

A project manager usually faces with the problem of estimating the effort needed to develop a project. This task is obviously dependent on the software engineers in the team but it can be measured with different techniques. Some of these techniques are easy to use but require lots of additional data while the others are time consuming and difficult to follow (Radlinski & Hoffmann 2010). So it can be said that there is no simple way to make an accurate estimate of the effort required to develop a software system (Sommerville 2011).

Even you make initial estimations; there are several drawbacks in effort estimation like the software engineers involved in the project and their ability. But still organizations need to make effort estimation. And usually all these estimators are done to estimate the effort for a brand new project by project managers who use their experience-based judgments and knowledge from previous projects.

First let's explain what Effort is. Effort is the amount of time that is spent to finish a project, expressed in terms of man-month or person-hours. For evaluating the software cost estimation there are three main factors: effort, calendar time and the total cost. By far, finding the effort is the key point for software cost estimation since it will affect both the duration and the fund needed for the project. Until today, for cost estimation different techniques like have been used: expert judgment, analogy-based estimation, algorithmic models, machine learning (Elish 2009; Nayebi et al. 2015; Humayun & Gang 2012), proxy-based estimation, and the use of historical data (Nayebi et al. 2015).

Expert Judgment is a term that refers specifically to a technique in which judgment is made based upon a specific set of criteria or expertise in a person in a specific area on similar projects, the accuracy for this technique highly depends on the ability and experience of the expert who examines the new project (Elish 2009).

Analogy-based estimation is a term that refers to a technique used for estimation based on historical data for an analogous system or subsystem. In this technique, a currently fielded system, similar in design and operation to the proposed system, is used as a basis for the analogy.

Proxy-based estimation is a technique that estimates the size of a program by applying historical size data to a conceptual design, and then applying statistical techniques to adjust the estimate based upon past estimating accuracy (Schoedel 2006).

The aim of this work is to estimate software effort objectively by using machine learning techniques instead of using subjective and time consuming estimation methods like expert judgment and estimation by analogy. To add more, Unsuitable criteria and unsuitable technique for estimation may be chosen by an expert. For these reasons, it is highly advantageous to use a more structured estimation process using machine learning techniques (Nayebi et al. 2015).

There are many advantages for machine learning including: (Elish 2009; Prabhakar 2013; Idri et al. 2002)

a. The capability to learn from past historical data.

b. The capability to model complex set of relationships between the effort (dependent variable) and the effort drivers (independent variables).

1.2 OUTLINE

This research is organized as follows:

Section 1 in this work is the introduction part, which also contains the outline. Section 2 is about related work.

In Section 3, Data and Methodology will be explained in detail including the dataset and machine learning techniques used in the experiments.

Section 4 describes the research procedure where as section 5 explain about the proposed solution.

Section 6 explains the results from all the experiments, and Section 7 will discuss the work and how it is done to find these results and the conclusion of this work and future work are also given.

2. LITERATURE REVIEW

To help the project managers decide on the effort, many software estimation models have been proposed such as SLIM, Checkpoint, Price-S, Seer and COCOMO (Boehm & Abts 1998; Baskeles et al. 2007). Instead of using expert knowledge alone which makes the decision process less subjective to their experience, it can be better to use the data collected with one of these models. Even if there is data collected, expert knowledge would still be needed to interpret it. Instead of this step, data obtained from one of these techniques can be analyzed with the help of machine learning techniques. In other words, machine learning techniques can be used to analyze the data collected from previous similar software projects and come up with an effort estimate. Therefore there are several studies on software effort prediction using machine learning techniques.

One of these studies aimed in finding the possibility to predict software development effort easily by using local data (Radlinski & Hoffmann 2010). They compared accuracy for 4 datasets (MAXWELL, DESHARNAIS, COCOMO, and QQ-DEFECTS) by applying 23 machine learning methods on them, they concluded that higher accuracy prediction occurred after applying feature selection, and also they found that project size (expressed in Kilo Line Of Codes (KLOC)) was the most influential factor, to add more, results also show that the accuracy of predictions for each technique varies depending on the dataset used (Prabhakar 2013).

While this study was concentrating on using local data, in another study Braga et al. (2007) tried to improve precision of effort estimation for software projects using 2 datasets (NASA and DESHARNAIS), they combined the machine learning techniques with robust confidence intervals that was computed from errors collected from the training sets after building regression model, this step was to improve the precision and reliability. As a result, Bagging showed improvement in performance for machine learning methods.

On the other hand, Baskeles et al. (2007) used two public datasets (USC and NASA) and a third dataset collected from Turkish software organizations, they applied several machine learning techniques to past data gathered using COCOMO guidelines based on

COCOMO and COCOMO2 models. All experiments were carried out by cross-validation. As a conclusion of this experiment, they have seen that parametric models are insufficient for software effort estimation, and the problem must be handled using an evolving system rather than a static one, they also found that it is highly preferable to use a learning system instead of a parametric model.

Mair et al. (2000) applied three machine learning techniques: Artificial Neural Networks (ANN), Case-based Reasoning (CBR) and Rule Induction (RI) with a Least Squares Regression procedure (to provide a benchmark comparison) to DESHARNAIS dataset, in order to compare effort prediction systems in terms of three factors: accuracy, explanatory values and configurability. As a result from this research, they found that ANN technique has the most accurate prediction, but they argued that there are other factors influencing the prediction, so they found that the explanatory value of CBR and RI estimation gives them an advantage when considering their interaction with human involvement.

Elish (2009) evaluated the potential of multiple additive regression trees (MART) as a novel software effort estimation model when compared with other recent published models, in terms of accuracy. MART extends and improves the classification and regression trees (CART) model when using stochastic gradient boosting. He made his experiments using NASA dataset. Results show that an improvement in accuracy estimation of software project effort by using MART when compared to linear regression, Radial Basis Function (RBF), and Support Vector Regression (SVR).

Another experiment was done by Prabhakar (2013) who used two machine learning techniques (ANN and Support Vector Machine (SVM)) to predict software effort using China dataset, he developed MATLAB programs for training and testing. He concluded that ANN with one hidden layer and SVM with ANOVA kernel show the best results, also the earlier technique (ANN) has outperformed the later (SVM).

Nayebi et al. (2015) proposed a way to find a model for effort estimation that has the most accurate and certain estimated for a dataset, depending on prediction, correlation coefficients and Bayesian information criterion. They applied 9 machine learning methods to 9 different datasets, they concluded that M5-Pruned (M5P) decision tree

algorithm and linear regression with feature subset selection outperformed other models, also results show that feature selection make model performance better than before.

In another work, MacDonell & Shepperd (2003) tried to find if it is possible to combine some effort prediction techniques in order to optimize the effort in software projects, the prediction techniques used in this research are: Expert judgment, Least-squares linear regression(LSR) and CBR. They used a dataset of 77 observations from a medical records system within five months. The dataset was divided into a ratio of 2:1 which describes the training and testing sets respectively. They found that indeed there are potential advantages when using more than one technique for effort prediction, because sometimes a technique can show poor prediction, but they couldn't find which technique must be used first.

Shivhare & Rath (2014) applied two machine learning techniques (Naïve Bayes Classifier and ANN) on a public dataset (USP05) to predict software effort after performing feature reduction to the original dataset using rough-set analysis, They concluded that Naïve Bayes classifier performs better.

Also, many reviews and surveys have been made by many researchers in the field of Cost and Effort Estimation and Machine Learning Techniques including (Batra & Barua 2013; Wen et al. 2012; Singh et al. 2007; Boehm & Abts 1998; Ferrucci et al. 2010; Fedotova et al. 2013; Khatibi & Jawawi 2010; Molokken & Jorgensen 2003; Rodríguez-Soria 2010; Khan & Qureshi 2014). Most of them concluded that effort should be estimated in the early stages in the software development life cycle, exactly in the analysis and design phases, to know the approximate staff to hire, also to know the time and cost needed to finish the project.

The difference between all the mentioned studies and our study is the use of machine learning techniques specific for classification problems and combining them together using a special technique for improving the accuracy that will be mentioned about and discussed in section 5.

Effort Estimations should be both accurate and certain (Nayebi et al. 2015). Researchers have been trying to build models for estimations since the 1960s

(Jørgensen 2007; Nayebi et al. 2015). All the studies that I have mentioned above are concerning about accuracy of the prediction, all the researchers are trying to build a model that will increase the accuracy and decrease uncertainty for the software development effort estimation.

A comparative analysis has been done by Humayun & Gang (2012) by comparing traditional techniques (Expert Judgment and Algorithmic Estimation) and the use of machine learning techniques. They discussed many machine learning techniques that have been used in effort estimation problems including (ANN, CBR, RI, GA, CART, and MART). They concluded that machine learning give more accurate estimation compared by traditional techniques.

In the following sections and subsections, we are going to discuss each and every technique we have used in our experiments, the datasets that we have used to train our model, all the steps that have been taken to make the experiments, and at last, the results for all the estimations and calculations will be shown and discussed.

3. DATA AND METHODS

The engineering discipline which cover all the aspects of software production is Software Engineering (Sommerville 2007). The software product here should not be considered as the developed computer program but also the products like the configuration files, system documentation and user documentation.

It is a very great challenge for a software company to develop a new software project in a high quality within predetermined budget and time (Prakash et al. 2013). Effort estimation is the first step that is taken in budgeting for the new software project.

This research was prepared to predict effort for new software development projects by studying and analyzing previous data combined from earlier software projects. Data that is exploited from earlier projects consists of the actual effort (the dependent attribute) and also factor values that are related to the effort (the independent attributes).

3.1 DATASETS USED

Many datasets have been used to estimate software development effort. The mostly used software datasets to predict software effort are: China (Prabhakar 2013), Maxwell (Radlinski & Hoffmann 2010), NASA (Baskeles et al. 2007; Braga et al. 2007; Elish 2009; Oliveira 2006), Finnish, Telecom, USC (Baskeles et al. 2007), CoCoMo (Boehm 1981; Radlinski & Hoffmann 2010), Kemerer, ISBSG, Albrecht, Miyazaki94 and Desharnais (Radlinski & Hoffmann 2010; Mair et al. 2000; Braga et al. 2007; Burgess et al. 2001). Many of these datasets are publicly available in PROMISE repository which is one of the most famous used repositories in Software Engineering Community to estimate effort, it is a well-known, useful and real set related to projects of software engineering, made publicly available in order to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering (Sayyad Shirabad, J. and Menzies 2005). These datasets have been constructed and developed by various companies, some of them are cross-company and the others are single-company related projects.

In this research we are going to use two publicly available datasets, Desharnais and Maxwell, in order to build a model for estimating the effort for new software development projects.

3.1.1 Desharnais Dataset

In this work, to find the potentiality of machine learning techniques for building software development effort prediction, we have used Desharnais dataset; it is one of the most commonly used datasets in the field of software effort estimation. Many Researchers used this dataset in their experiments including Radlinski & Hoffmann (2010); Braga et al. (2007); Mair et al. (2000) and many others. Desharnais dataset consists of 81 projects collected by J.M. Desharnais in the late 1980s from a Canadian software house (Mair et al. 2000; Desharnais 1989; Menzies et al. 2012). The original dataset consists of 12 attributes but in this study the ProjectID attribute was omitted from the original dataset because it has no meaning to the study, so the left 11 attributes are: (TeamExp, ManagerExp, YearEnd, Length, Effort, Transactions, Entities, PointsAdjust, Adjustment, PointsNonAjust, Language) as explained in Appendix-1, ten independent attributes and one dependent attribute (effort), all the values in this dataset are numeric but only one nominal attribute that is Language. Despite the fact that this dataset is now more than 25 years old, it is one of the largest and most used publicly available datasets (Mair et al. 2000).

3.1.2 Maxwell Dataset

Maxwell dataset is a new dataset consists of 62 projects (Maxwell 2002). Each project is described by 27 attributes in which all attributes are numerical. 26 independent attributes and one dependent attribute (effort). As explained in Appendix-2.

The two mentioned datasets (Desharnais and Maxwell) have some common features and other different features, for example, they both have the same type of project size which is described using function points (FP), and also they both have the same measure type for effort which is person-hours. Table 3.1 below summarizes the datasets in term of the features and projects within each dataset.

Table 3.1: Summary of the datasets

Dataset	Number of Features	Number of Projects
Desharnais	12	81
Maxwell	27	62

Source: (Maxwell 2002; Desharnais 1989)

For the two datasets mentioned above, we divided the dataset into two parts, training set and testing set. To form these parts, we randomly divided the datasets using two techniques which are Leave-One-Out Cross Validation Technique and K-Fold Cross-Validation Technique, which will be discussed later in detail.

The two datasets are analyzed in their own context by using two machine learning techniques which are *K*-nearest neighbor (*k*-NN) and Support Vector Machine (SVM) that will be discussed later in the following subsection.

3.2 MACHINE LEARNING TECHNIQUES USED

Machine learning is considered as a subfield of Artificial Intelligence and it is concerned with the development of methods and techniques that enable the machine to learn and perform activities and tasks (Prabhakar 2013). Machine Learning techniques resemble the human mind in some aspects, allowing us to solve complex problems in a fast way (Schank 1982). Recently, Machine Learning approaches have been proposed as an alternative way to predict software effort (Mair et al. 2000).

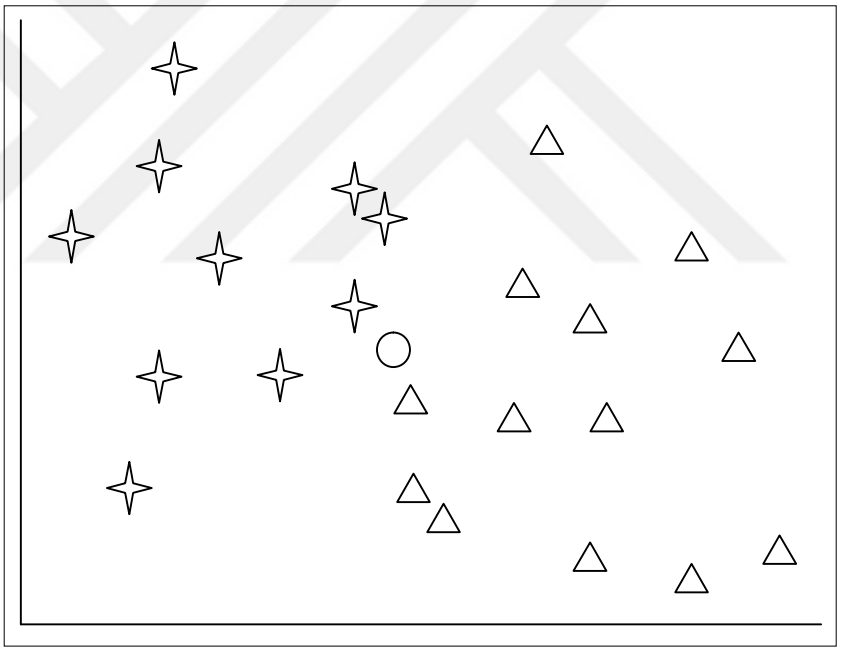
In this section, we will discuss two Machine Learning techniques that could be used to predict effort: *k*-NN and SVM. These techniques have been selected because *k*-NN is non-parametric and SVM is parametric and this difference will enable us to reveal what kind of a model works better on software effort estimation datasets. The main difference between a non-parametric and parametric model is the number of parameters and how they are obtained (Murphy 2012). In a non-parametric model the parameters are detected by the training data itself but not the model. However, in a parametric model the parameters are determined by the model itself. That's why the number of parameters increases as the number of training instances increase in a dataset in a non-parametric model. Nonetheless this is not valid for a parametric model since the number of parameters is fixed and is not dependent on the number of training instances.

3.2.1 *k*-Nearest Neighbor

k-NN is one of the techniques used for classification problems, and it is one of the most simple classification techniques that should be the first option for a classification study when there is no past knowledge about data description (Peterson 2009). *k*-NN works first by computing distance between an instance with other instances and find the *k* nearest neighbor for that instance, then it estimates the effort (Nayebi et al. 2015).

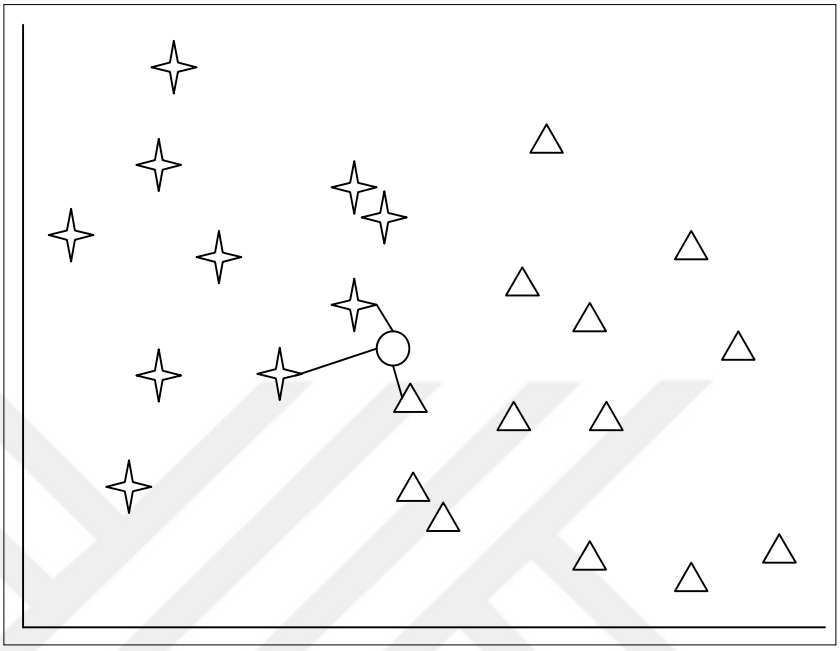
Let's describe this non parametric classifier with figures. As it is seen in Figure 3.1 samples belonging to two different classes are represented by stars and triangles, and the unclassified sample is shown with a circle.

Figure 3.1: Finding the class of a sample with k-NN



For the neighborhood parameter $k = 3$ (for example), first the nearest three samples are found, as shown in figure 3.2. The distance between these samples are usually calculated by Euclidean distance (default in MATLAB) which is the strait-line distance between two points in Euclidean space. The majority of the samples that are belonging to one class determine the label of the new sample.

Figure 3.2: Calculating the distance between new sample and the new one



Since the nearest two samples in this example are from the class represented by star, the new sample is also labeled as star, as shown in figure 3.3.

Figure 3.3: Deciding on the class of the new sample

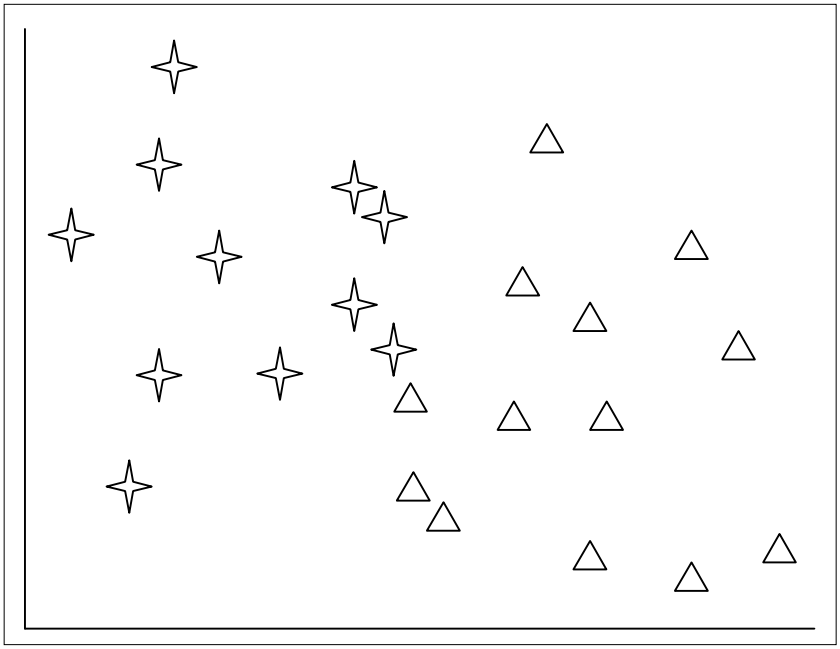
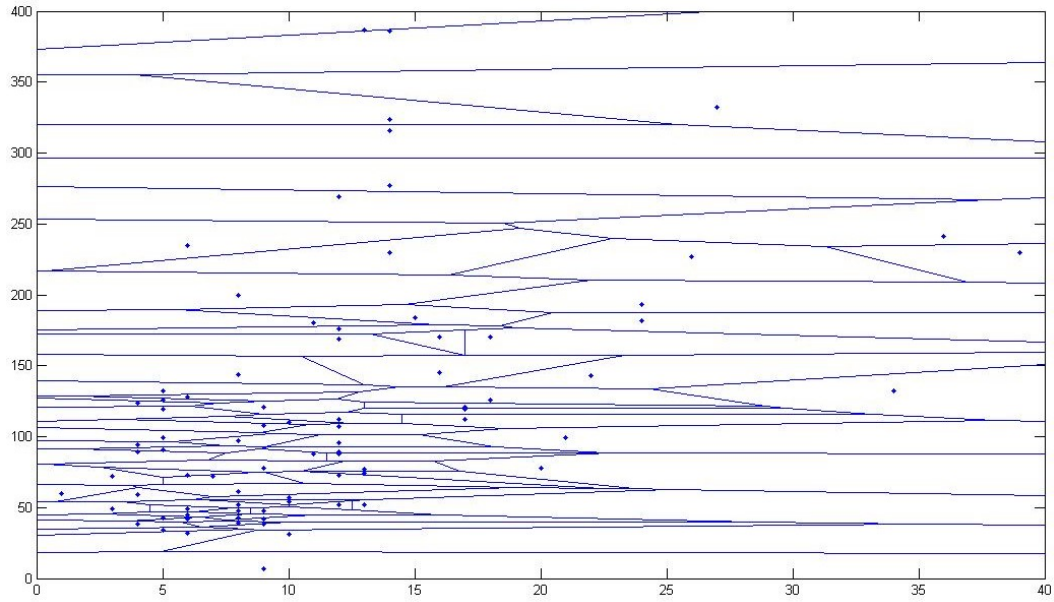


Figure 3.4 shows a visualization for *k*-NN using Desharnais dataset which we used to train our model, it is made by using a ready function in MATLAB called (voronoi).

Figure 3.4: Voronoi graph for k-NN visualisation



3.2.2 Support Vector Machine

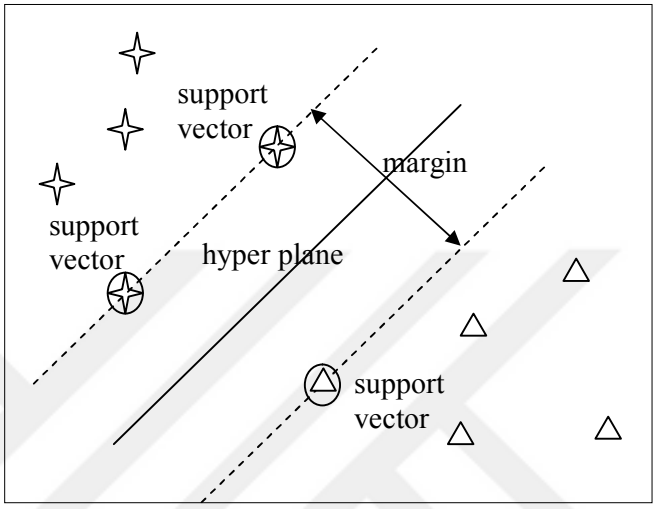
SVMs are a set of machine learning methods used in many areas, such as classification and regression (Smola & Schölkopf 2004). This method has outperformed previous ones in many classification and regression tasks.

SVM algorithm was proposed by Vapnik (in Cortes & Vapnik 1995) and got the attention of the researchers working in machine learning. SVM classifier separates the instances from two different classes by using a hyper plane which tries to maximize the margin (Vapnik 2013). This increases the generalization capability of the classifier. The instances that are close to or on the border are called the support vectors. The number of support vectors also represents the complexity of the model. A figurative representation of the algorithm is a good way to visualize how it works, as shown in figure 3.5 below.

It is easy to use the support vector machine algorithm by taking some important issues into consideration (Vapnik 1998). First important issue is to decide on the place of the hyper plane that will separate the instances. Here support vectors have direct influence

on the optimum location of this decision surface since they are the data points most difficult to classify.

Figure 3.5: Support vector machine algorithm (linear kernel)



In machine learning, if a problem is not linearly separable, one can try to fit a non-linear function. However, non-linear functions are time consuming and more difficult to understand. In this case the problem can be mapped to a new space by using nonlinear basis functions. Of course, this will increase the dimensionality of the problem. So it is better to find a model whose complexity is not dependent on the input dimensionality. And SVMs are one of the most popular algorithms that are running on this trick.

When the input space is carried to another dimension, a significant point is to map the items in the original input space to the new ones; this is done by the kernel function.

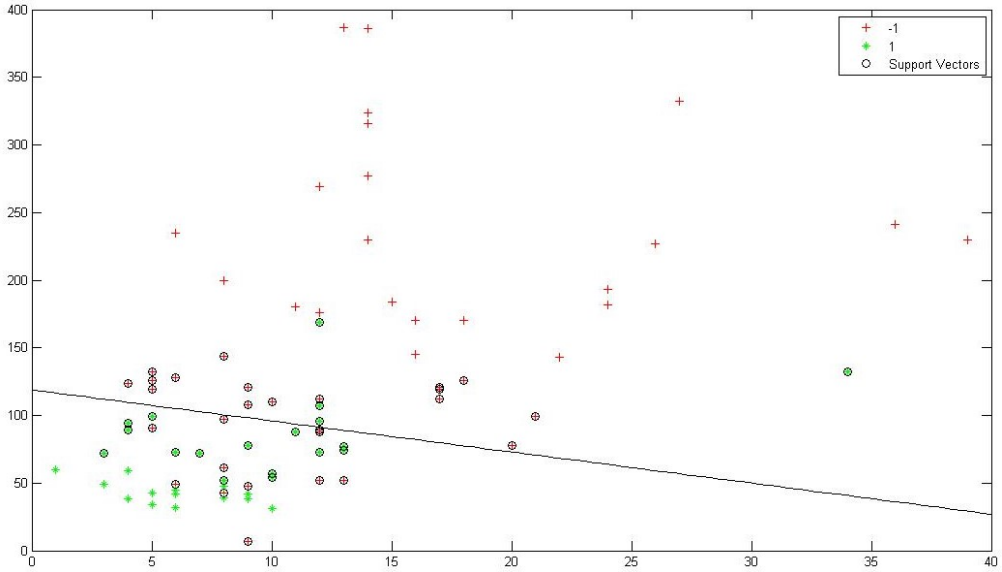
The last important issue is the over-learning of the classifier. Over-learning should be avoided to make a more generalizable model. By using different kernels even one hundred percent classification can be obtained. But the model complexity is as important as the classification accuracy. And to control the complexity of SVM, cost parameter is used.

So when using the SVM classifier, main three parameters are used (1) the kernel type (linear, quadratic, polynomial, RBF, etc.), (2) the cost parameter / box constraints, and

(3) the kernel width. The values of these parameters change according to the problem and usually determined empirically.

Figure 3.6 illustrates visualization for the SVM technique when applying it to Desharnais dataset.

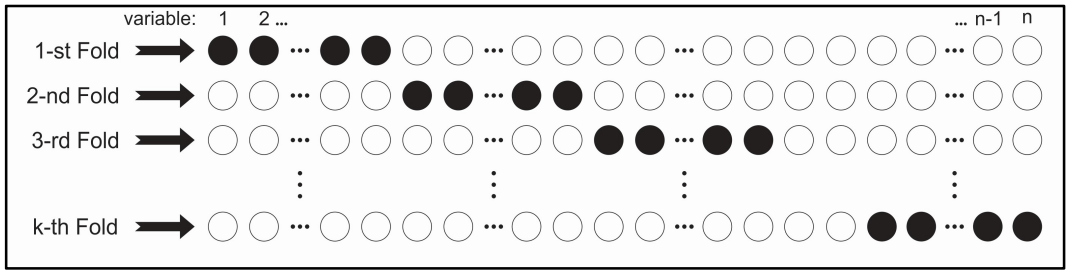
Figure 3.6: visualization of SVM using 2 classes in Desharnais dataset



3.2.3 K-Fold Cross Validation

K-Fold Cross Validation works by dividing the dataset into K subsets, K-1 subsets are used for training and the final subset is used for testing, this process is repeated K time by leaving out one different subset for testing each time, then the mean value is measured. Different values of K can be used in K-Fold Cross Validation technique; in our research we used the values (3 and 10), we found that 10-Fold cross validation is better than 3-Fold cross validation.

Figure 3.7: K-Fold cross-validation technique

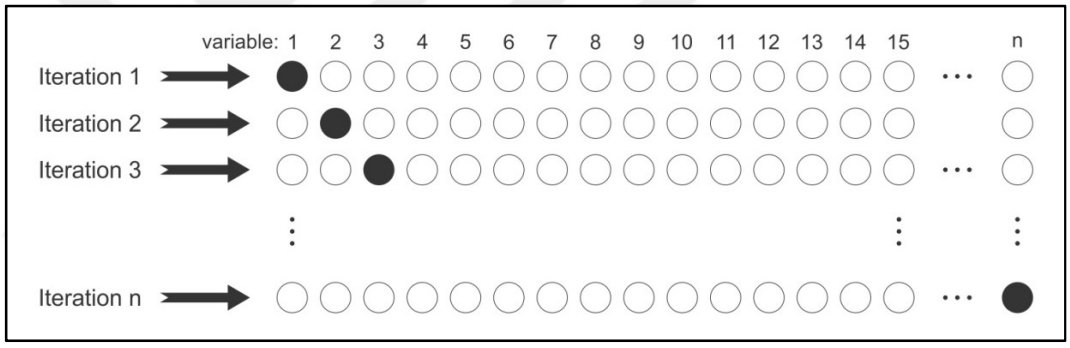


In figure 3.7 above, the black circles describe the testing sets and the white circles describe the training sets, the dataset is divided into k folds and in each fold we have training and testing sets and we find the prediction for every variable in each fold, then the total accuracy is calculated.

3.2.4 Leave-One-Out Cross Validation

This technique works by leaving one sample as testing set and all the other projects were considered as the training set, this procedure works fine because we have relatively small datasets (one of them has 62 projects and the other has 81 projects). This technique was seen to be more effective and produces more accurate estimations.

Figure 3.8: Leave-One-Out cross-validation technique



In figure 3.8 above, black circle in each iteration describes the testing set and all the white circles describe the training set. In this technique, the dataset is divided into n iterations according to the number of variables, and in each iteration the n^{th} variable is considered the testing set, then the total accuracy is calculated.

4. RESEARCH PROCEDURE

This research aims to improve software development effort estimation by developing a model and using different machine learning techniques on software development effort estimation public datasets.

We begin by taking into account some machine learning techniques for estimation, comparing their performance on a particular dataset, and resulting to a set of standards and criteria for selecting the model with the best performance on the dataset.

This procedure consists of many steps that will be discussed in detail in the following subsections:

4.1 CHOOSING THE DATASET

Choosing the best dataset that will work on classification, so we found 3 datasets that have numerical values, these are Desharnais, Maxwell and China, but we will work only using 2 of them (Desharnais and Maxwell).

We used WEKA software to convert the original file for the dataset that was in .arff format, and changed it to .xls format, to be able to use it in MATLAB.

4.2 WORKING ON THE DATASET

This step covers transforming data into a new scales, this procedure is called Discretisation, to change the problem from regression to classification (i.e. to discretize numeric variables to a number of classes/intervals), it is a very important step to be done because the machine learning techniques we have picked work only with categorized data and do not predict exact numeric values but one of the classes, we used a code in MATLAB to convert the actual effort taken from the dataset to a specific number or label (1 to 5) according to an experiment done by Radlinski & Hoffmann (2010), as explained in Tables 4.1 and 4.2 below.

Table 4.1: Defining intervals for Effort in Desharnais dataset

Desharnais Dataset		
Actual Effort	Number of Cases	Class ID
0 – 1500	11	1
1500 – 3000	21	2
3000 – 4500	20	3
4500 – 8000	14	4
More than 8000	15	5

Source: (Radlinski & Hoffmann 2010)

Table 4.2: Defining intervals for Effort in Maxwell dataset

Maxwell Dataset		
Actual Effort	Number of Cases	Class ID
0 – 1500	9	1
1500 – 3000	10	2
3000 – 5000	11	3
5000 – 10000	18	4
More than 10000	14	5

Source: (Radlinski & Hoffmann 2010)

4.3 FILLING MISSING VALUES

The third step is about filling missing values in the datasets, we found that Desharnais dataset has 4 projects containing missing values; we calculated the mean value for all the numeric variables in one attribute (column) and filled the missing value.

4.4 AUTO-SCALING

Next we applied Auto-scaling on the whole dataset after filling the missing values, this method uses mean-centering followed by division of each column by the standard deviation of that column. We used a ready function in MATLAB called (autosc) that will return a scaled dataset, mean value (mx) and standard deviation (stdx) for the dataset.

4.5 CROSS VALIDATION

This step is about dividing the dataset into training and testing sets, two machine learning techniques have been used for this purpose, which are Leave-One-Out and K-Fold cross validation techniques.

4.6 ESTIMATIONS

In this step, we applied the machine learning techniques, which are *k*-NN and SVM, using 10-Fold and leave-one-out cross validation techniques to generate predictions.

4.7 SELECTION CRITERIA

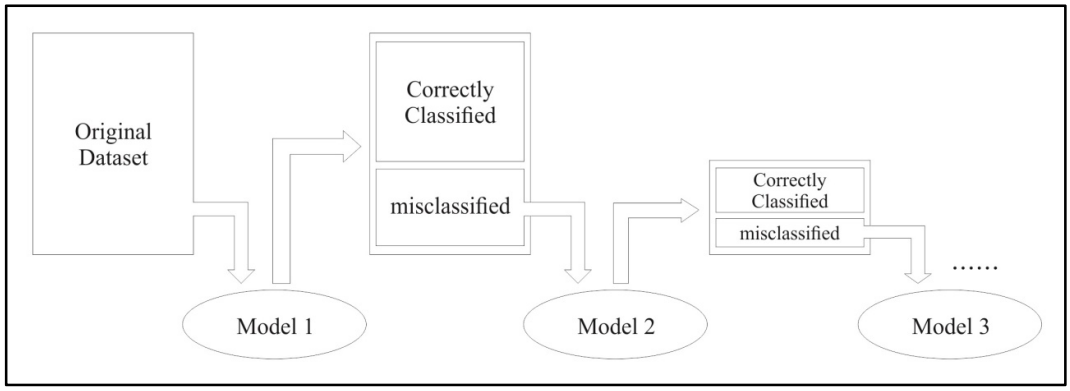
In this study, we used only one criterion, which is Accuracy. The accuracy is implemented for each model on each dataset separately.

5. PROPOSED SOLUTION

The aim of this research is to improve the accuracy of estimations for software development projects, this will be done by constructing a system that will use multiple machine learning techniques and apply them to multiple datasets. In this research, we are going to propose a method to improve accuracy for effort estimation on software development projects, this method is called Boosting, it is widely used to improve the accuracy for any learning algorithm, boosting is a method that is used to boost accuracy of any learning algorithm by fitting a series of models each having low error rate and then combining them into an ensemble that may perform better (Monteiro 2002; Schapire et al. 1999; Elish 2009).

Many algorithms have been used for the purpose of boosting, and one of them is called Adaptive Boosting (Known as AdaBoost). AdaBoost algorithm was first introduce by Freund & Schapire (1997), this algorithm was a solution to many of the difficulties for earlier boosting algorithms (Schapire 2009), the idea of AdaBoost is to construct a strong model sequentially by combining multiple weak classifiers into one single strong classifier, a weak classifier is a classifier which perform poorly but better than random guessing. Figure 5.1 below illustrate this procedure.

Figure 5.1: How Adaptive Boosting works



Here, each model tries to correct the mistakes of the previous one, to come up with a better accuracy for effort estimation. AdaBoost can be applied to any classification algorithm.

AdaBoost was used by many researchers in their experiments including Reyzin (2003); Schapire (2013); Kummer & Najjaran (2014); Schapire (2009). The boosting algorithm AdaBoost is explained in figure 5.2 below.

Figure 5.2: AdaBoost algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t} \left[h_t(x_i) \neq y_i \right].$$

- Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$.
- Update:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

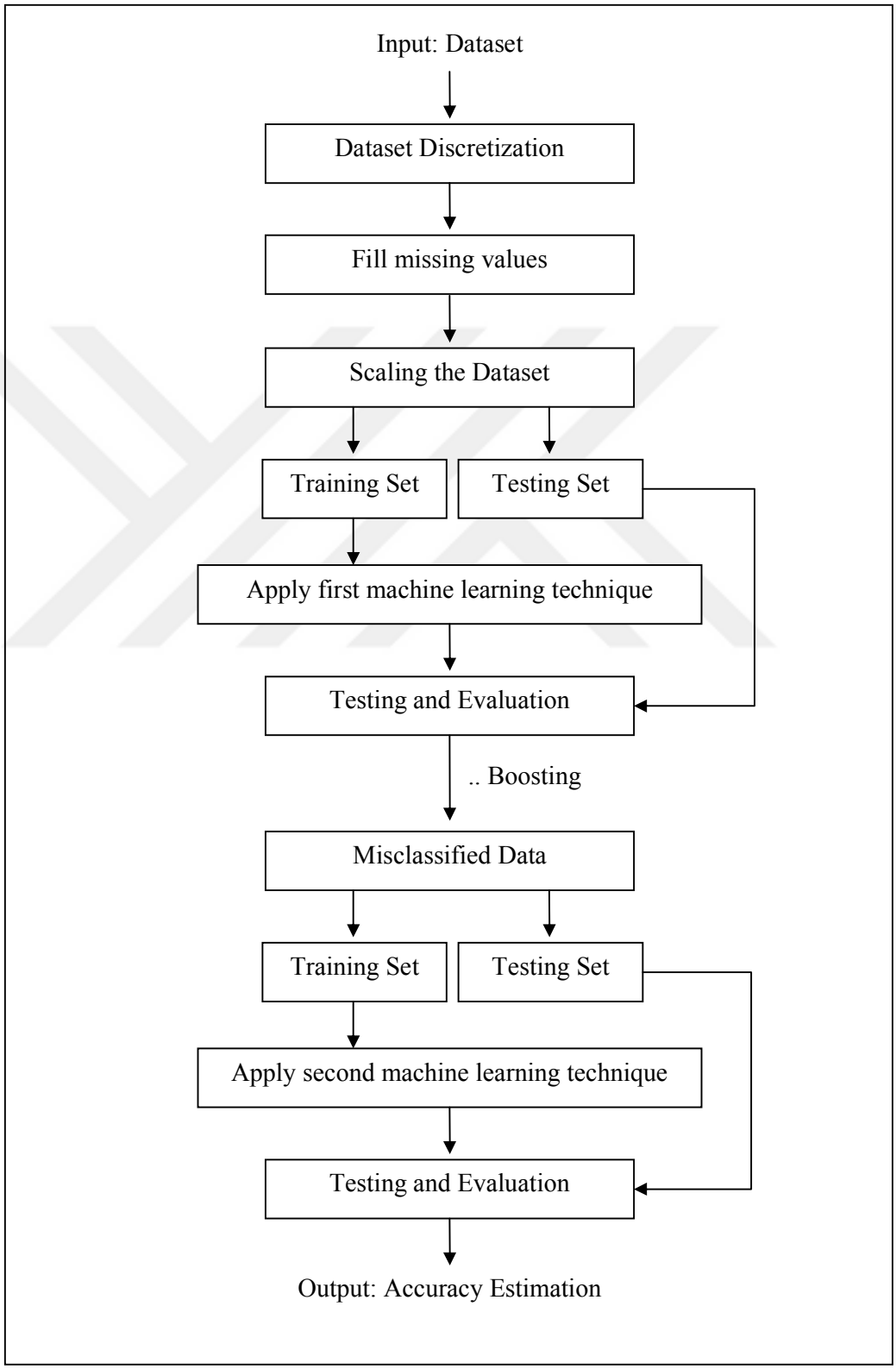
Output the final hypothesis:

$$H(x) = \operatorname{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right).$$

Source: (Schapire 2009)

Figure 5.3 below illustrates all the steps, after calling the dataset and inputting it to the system; discretization of the variables, where all the actual effort values were replaced into classes between 1 and 5, because we are going to use machine learning techniques for classification, then the dataset is checked for missing values in its variables, if there is a missing value, it is replaced with a value by computing the mean value for all the variables in the same attribute. After this step has been completed, the dataset is scaled and divided into training and testing sets, this step has been done by using two techniques (Leave-one-out and K-fold Cross Validation), and then the first machine learning technique is applied to train the system, at last, the evaluation of the system performance using the testing set and comparing the results.

Figure 5.3: Visualization of our proposed solution



After getting the results from the first technique, the misclassified data are transformed to another dataset; the new dataset is also divided into two sets like the previous one and then a second machine learning techniques is applied for training. Final step is about calculating the overall accuracy after implementing AdaBoost technique.

6. EXPERIMENTAL RESULTS

In this section, we will present results for all the experiments we have done using the machine learning techniques on the mentioned datasets.

All the experiments was conducted using a recent version of MATLAB software (R2014b), MATLAB is known to be simple, easy to use and has many ready functions built in its library.

Two publicly available datasets have been used in the experiments, those are Desharnais and Maxwell. The two datasets were divided into two sets, training set and testing set. We used two approaches to divide the dataset, firstly by using Leave-One-Out Cross-Validation, and the other by using K-Fold Cross-Validation.

All the experiments were conducted two times, one without scaling the dataset, and then by scaling the dataset using a ready function in MATLAB called (autosc). Results show that autoscaling improves the accuracy when implementing it to *k*-NN technique, but it does not improve it when using SVM technique. The following sub-sections explain all the results for the mentioned experiments.

6.1 RESULTS FOR DESHARNAIS DATASET

As explained earlier, Desharnais dataset consists of 81 software projects between 1983 and 1988, it has 4 projects missing some values, but we filled them by measuring the mean value for all the values in the same attribute, after filling all the missing values, we applied two machine learning techniques to calculate the accuracy and find the best one, the two machine learning techniques are *k*-NN and SVM. All the results are shown in the following sub-sections:

6.1.1 *k*-NN Technique on Desharnais Dataset

k-NN is simple and easy to use technique in which it is used for classification problems like the one we have. It works by computing the distance between an unknown instance (the one which we want to predict) and other known instances (the original values in the dataset), the new value for the unknown instance will take the value of the majority of the surrounding instances.

As we mentioned before, we applied Discretization to the values in the dataset, to change the problem from regression to classification problem (i.e. we removed the actual effort and changed it to labels/ classes from 1 to 5), so the problem now is a classification problem with 5 classes, we applied K-NN techniques and the results are shown in Table 6.1 below:

Table 6.1: Using k-NN technique on Desharnais dataset

Leave-One-Out Cross-Validation	Accuracy without Autoscaling	Accuracy with Autoscaling
k=1	40.74	43.21
k=3	41.98	41.98
k=5	40.74	43.21
k=7	45.68	**50.62**
10-Fold Cross-Validation	Accuracy without Autoscaling	Accuracy with Autoscaling
k=1	27.16	19.75
k=3	27.16	23.46
k=5	28.4	23.46
k=7	**29.63**	23.46

We can see from the table above that the best accuracy is when using 7 nearest neighbors and with scaling the whole dataset, this is when applying Leave-One-Out Cross-Validation, this accuracy is relatively small, but it is the best accuracy in this situation.

6.1.2 SVM Technique on Desharnais Dataset

Accuracy was compared by changing the kernel function (Linear, Quadratic, Polynomial, and RBF) and the box constraint (between 0.01 and 1). Experiments were carried out two times, first without scaling the dataset and the second time with auto scaling the whole dataset.

As it is mentioned before, data was transformed into a new scale (Discretisation), to change the problem from regression to classification, the previous technique (K-NN)

worked fine with 5 classes, but SVM will only work with 2 classes, so we made this step in many ways to make sure we get the best accuracy:

First way was only by producing 2 classes (1 or -1), one of them when the effort is 3000 or less and the other class when the effort is more than 3000; results are shown in table 6.2 below.

Table 6.2: Results for SVM technique on Desharnais dataset when using 2 classes

Kernel Function	Accuracy without Autoscaling	Box Constraint	Accuracy with Autoscaling	Box Constraint
Linear	79.01	1	79.01	1
Quadratic	81.48	1	81.48	1
Polynomial	**85.18**	**1**	**85.18**	**1**
RBF	79.01 (sigma=1)	1	79.01 (sigma=1)	1

Second way was by producing 5 classes and then comparing each class with all the other classes together, and then taking the total average of the accuracy, for example, the first class will be 1 and all the other classes will become -1 and we find the accuracy, then the second class will be 1 and all the other classes will become -1 and we find the accuracy, and so on, then we find the total accuracy by taking the average value of all the five accuracies we have calculated, all the results are listed in table 6.3 below.

The earliest used implementation for SVM multi-class classification is probably the one-against-all method. It constructs k SVM models where k is the number of classes. The m^{th} SVM is trained with all of the examples in the m^{th} class with positive labels, and all other examples with negative labels.

Results show that RBF kernel function has the best accuracy with box constraint equal to 0.1 and the default value of RBF sigma (kernel width), this accuracy is the best accuracy for this experiment when applying the SVM machine learning technique alone, and it is clear from the results that Autoscaling didn't affect the results at all.

Table 6.3: Results for SVM technique on Desharnais dataset when using 5 classes

Kernel Function	Accuracy without Autoscaling	Box Constraint	Accuracy with Autoscaling	Box Constraint
Linear	65.92	1	65.92	1
Quadratic	76.29	1	76.29	1
Polynomial	76.05	0.01	76.05	0.01
RBF	**81.97** (sigma=1)	**0.1**	**81.97** (sigma=1)	**0.1**

The third way was by producing 5 classes and then comparing two classes together, in table 6.4, we compared the first and second classes, and in table 6.5 we compared the second and third classes together.

Table 6.4: Results for SVM technique on Desharnais dataset when using 5 classes with comparing the first and second classes

Kernel Function	Accuracy without Autoscaling	Box Constraint	Accuracy with Autoscaling	Box Constraint
Linear	62.50	0.01	62.50	0.01
Quadratic	**75.00**	**0.1**	**75.00**	**0.1**
Polynomial	68.75	0.01	68.75	0.01
RBF	65.62 (sigma=0.1)	0.1	65.62 (sigma=0.1)	0.1

Numbers show that the best result when using 5 classes and comparing the first two classes is when using Quadratic kernel function when and a box constraint equal to 0.1, but when comparing the second and third classes, the best results are when using Quadratic and RBF kernel function with the default value of kernel width.

Table 6.5: Results for SVM technique on Desharnais dataset when using 5 classes with comparing the second and third classes

Kernel Function	Accuracy without Autoscaling	Box Constraint	Accuracy with Autoscaling	Box Constraint
Linear	63.41	0.01	63.41	0.01
Quadratic	**70.73**	**0.1**	**70.73**	**0.1**
Polynomial	68.29	1	68.29	1
RBF	**70.73** (sigma=1)	**0.1**	**70.73** (sigma=1)	**0.1**

6.1.3 Boosting Results on Desharnais Dataset

Boosting was done by using one technique and calculating the accuracy, then taking the misclassified data from the first technique into a new dataset and using another technique. Here we applied this method in two ways, first by using K-NN and boosting it with SVM, the other by using SVM and boosting it with K-NN, all the results are shown in tables 6.6 and 6.7 below.

Table 6.6: Boosting K-NN with SVM on Desharnais dataset

K-NN then SVM	Without Autoscaling	With Autoscaling
Accuracy for K-NN	45.68	50.61
Accuracy for SVM	70.45	80.00
Total Accuracy	83.95	**90.12**

Table 6.7: Boosting SVM with K-NN on Desharnais dataset

SVM then K-NN	Without Autoscaling	With Autoscaling
Accuracy for SVM	85.18	85.18
Accuracy for K-NN	41.66	41.66
Total Accuracy	91.35	**91.35**

We can observe from the results above mentioned in tables 6.6 and 6.7 that boosting really improves the accuracy, at least it is approximately 84% (this is when using K-NN first) and at most it is 91.35% (this is when using SVM first), this result is very good

and show that the best way to improve accuracy is to use adaptive boosting as shown in all the previous experiments.

6.2 RESULTS FOR MAXWELL DATASET

Maxwell dataset is relatively new and small dataset which consists of only 62 projects between 1985 and 1993; it has no missing values so it is not necessary to apply a code to fill missing values in MATLAB like the previous dataset (Desharnais), this is a very important thing to do to save time and memory usage when making the experiment.

Like the previous dataset, we applied two machine learning techniques (*k*-NN and SVM) on the dataset to find the best accuracy for the prediction; all the results are shown in the following sub-sections:

6.2.1 *k*-NN Technique on Maxwell Dataset

Table 6.8 show all the results when applying K-NN technique to Maxwell dataset, this procedure was implemented to times, first without Autoscaling the dataset, and the second time with Autoscaling the dataset using a ready function in MATLAB.

Table 6.8: Using k-NN technique on Maxwell dataset

Leave-One-Out Cross-Validation	Accuracy without Autoscaling	Accuracy with Autoscaling
k=1	41.93	32.25
k=3	**50.00**	38.71
k=5	46.77	32.64
k=7	48.38	35.48
10-Fold Cross-Validation	Accuracy without Autoscaling	Accuracy with Autoscaling
k=1	14.51	22.58
k=3	14.51	22.58
k=5	14.51	27.42
k=7	16.13	**32.25**

We can see from the table above that the best accuracy is not more than 50%; this is when performing Leave-One-Out Cross Validation using 3 nearest neighbors before auto scaling the dataset.

6.2.2 SVM Technique on Maxwell Dataset

SVM technique was done like the previous dataset but with changing the actual effort numbers like the experiment done by (Radlinski & Hoffmann 2010), results are shown in tables 6.9 to 6.12 below:

From reading the results in table 6.9, we can observe that the best result is when using a linear kernel function with the default values of box constraint and kernel width, this is when using only 2 classes (when the effort is 5000 or more, the class will be 1, and if it is less than 500, the class will be -1).

Table 6.9: Results for SVM technique on Maxwell dataset when using 2 classes

Kernel Function	Accuracy without Autoscaling	Box Constraint	Accuracy with Autoscaling	Box Constraint
Linear	**79.03**	**1**	**79.03**	**1**
Quadratic	62.90	1	62.90	1
Polynomial	70.97	1	70.97	1
RBF	51.61 (sigma=1)	0.1	51.61 (sigma=1)	0.1

But when using all the 5 classes, and comparing each class with all the other classes, the results will change, as shown in table 6.10 below.

Table 6.10: Results for SVM technique on Maxwell dataset when using 5 classes

Kernel Function	Accuracy without Autoscaling	Box Constraint	Accuracy with Autoscaling	Box Constraint
Linear	67.09	1	67.09	1
Quadratic	75.80	1	75.80	1
Polynomial	74.51	1	74.51	1

RBF	**80.00** (sigma=0.1)	**0.01**	**80.00** (sigma=0.1)	**0.01**

It is clearly shown in table 6.10 above that the best accuracy is 80% when using a RBF kernel function with box constraint equal to 0.01 and kernel width equal to 0.1; this is the best accuracy that we get when applying SVM technique to Maxwell dataset.

In the last way, we compared two classes together, in table 6.11 we compared the first two classes and we get 73.68% accuracy when using a linear kernel function with box constraint equal to 0.1. But in table 6.12, we compared the second and third classes, and we get 66.66% accuracy when using a quadratic kernel function with the default values of box constraint and kernel width.

Table 6.11: Results for SVM technique on Maxwell dataset when using 5 classes with comparing first and second classes

Kernel Function	Accuracy without Autoscaling	Box Constraint	Accuracy with Autoscaling	Box Constraint
Linear	**73.68**	**0.1**	68.42	0.1
Quadratic	47.37	1	47.37	1
Polynomial	42.10	1	42.10	1
RBF	52.63 (sigma=1)	1	52.63 (sigma=1)	1

Table 6.12: Results for SVM technique on Maxwell dataset when using 5 classes with comparing second and third classes

Kernel Function	Accuracy without Autoscaling	Box Constraint	Accuracy with Autoscaling	Box Constraint
Linear	52.38	0.1	52.38	0.1
Quadratic	**66.66**	**1**	57.14	1
Polynomial	42.85	1	47.62	1
RBF	42.85 (sigma=0.1)	0.1	52.38 (sigma=0.1)	0.1

6.2.3 Boosting Results on Maxwell Dataset

Boosting was done by using one technique and calculating the accuracy, then taking the misclassified data into a new dataset and using another technique. Here we applied this method in two ways, first by using K-NN then SVM, the other by using SVM then K-NN, all the results are shown in tables 6.13 and 6.14 below.

Table 6.13: Boosting K-NN with SVM on Maxwell dataset

K-NN then SVM	Without Autoscaling	With Autoscaling
Accuracy for K-NN	50.00	38.71
Accuracy for SVM	74.19	76.31
Total Accuracy	**87.10**	85.48

Table 6.14: : Boosting SVM with K-NN on Maxwell dataset

SVM then K-NN	Without Autoscaling	With Autoscaling
Accuracy for SVM	79.03	79.03
Accuracy for K-NN	30.77	15.38
Total Accuracy	85.48	82.25

As it is shown in the last two tables, results of accuracy is improving when using a boosting technique, the best accuracy was shown when applying K-NN technique first, we get 87 accuracy. The least accuracy is 82%, and it is more than the best accuracy when applying a machine learning technique alone. So we can say that adaptive boosting really improves the accuracy, and it is simple and easy to use.

7. DISCUSSION AND CONCLUSION

A survey conducted by (Molokken & Jorgensen 2003) show that approximately between 70 and 85 percent of the respondents accepted and agreed to the importance of estimating the effort for new software development projects. This study and experiment was done to evaluate some machine learning methods which are *k*-Nearest Neighbor and Support Vector Machine by applying them into two different datasets in order to make effort prediction for a new software development project.

We have seen from the results that when applying a single method alone, it has a good accuracy, but it did not get over 85% in the best scenario (this was when applying SVM technique using 2 classes to Desharnais dataset), but when we used a boosting technique called AdaBoost in order to improve the accuracy, we get 91.35% accuracy when using Desharnais dataset and 87.10% accuracy when using Maxwell dataset. So, we can say that boosting one technique with another one improves the accuracy of estimations.

For future work, other machine learning techniques can be used for classification problems, also other datasets can be used for experiments and training the model, in this way, we have more accurate estimations and predictions, also boosting with more than two techniques to get more accuracy reaching 100%. Another Future work can be to use feature selection technique.

APPENDICES

Appendix-1

List of Variables in Desharnais Dataset

Symbol	Name	Type
TeamExp	Team experience – measured in years	numeric
ManagerExp	Manager experience – measured in years	numeric
YearEnd	Year project ended	numeric
Entities	The number of entities in the systems data model (function points)	numeric
Transactions	A count of basic logical transactions in the system (function points)	numeric
Length	Actual project schedule in months	numeric
PointsNonAjust	Transactions + Entities (function points)	numeric
PointsAdjust	Function points adjusted by the Adjustment factor = 0.65 + (0.01 * PointsNonAdjust)	numeric
Adjustment	Function point complexity adjustment factor (Total Processing Complexity)	numeric
Effort	Actual Effort is measured in person-hours (Dependent)	numeric
Language	Programming Language	nominal

Appendix-2

List of Variables in Maxwell Dataset

Symbol	Name	Type
Syear	Software Year	numeric
App	Application Type	numeric
Har	Hardware Platform	numeric
Dba	Database	numeric
Ifc	User Interface	numeric
Sourse	Where Developed	numeric
Telonuse	Telon Use	numeric
Nlan	Number of Development Languages	numeric
T01	Customer Participation	numeric
T02	Development Environment Adequacy	numeric
T03	Staff Availability	numeric
T04	Standards Use	numeric
T05	Methods Use	numeric
T06	Tools Use	numeric
T07	Software's Logical Complexity	numeric
T08	Requirements Volatility	numeric
T09	Quality Requirements	numeric
T10	Efficiency Requirements	numeric
T11	Installation Requirements	numeric
T12	Staff Analysis Skills	numeric
T13	Staff Application Knowledge	numeric
T14	Staff Tool Skills	numeric
T15	Staff Team Skills	numeric
Duration	Duration (months)	numeric
Size	Application Size (Function Points)	numeric
Time	Time	numeric
Effort (H)	Work Carried-out	numeric

REFERENCES

Books

Desharnais, J.M., 1989. *Analyse statistique de la productivitie des projets informatique a partie de la technique des point des fonction.*

Maxwell, K., 2002. *Applied statistics for software managers*. Prentice Hall.

Menzies, T. et al., 2012. *The promise repository of empirical software engineering data*.

Murphy, K.P., 2012. *Machine Learning: A Probabilistic Perspective*. MIT Press.

Sommerville, I., 2011. *Software engineering*.

Sommerville, I., 2007. *Software Engineering. International computer science series*. Addison-Wesley.

Vapnik, V., 2013. *The Nature of Statistical Learning Theory* 2nd ed., Springer Science & Business Media.

Vapnik, V., 1998. *Statistical learning theory*.

Periodicals

Baskeles, B., Turhan, B. & Bener, a, 2007. Software effort estimation using machine learning methods. *Computer and information sciences, 2007. iscis 2007. 22nd international symposium*. pp.1–6.

Batra, G. & Barua, K., 2013. A Review on Cost and Effort Estimation Approach for Software Development. *International Journal of Engineering and Innovative Technology*. **3**(4), pp.290–293.

Boehm, B. & Abts, C., 1998. Software Development Cost Estimation Approaches – A Survey. *Science*.

Boehm, B.W., 1981. Understanding and controlling software costs.

Braga, P.L., Oliveira, A.L.I. & Meira, S.R.L., 2007. Software Effort Estimation using Machine Learning Techniques with Robust Confidence Intervals. *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, pp.352–357.

Burgess, C.J., Lefley, M. & Le, M., 2001. Can genetic programming improve software effort estimation? A comparative evaluation. *Information and Software Technology*. **43**(14), pp.863–873.

Cortes, C. & Vapnik, V., 1995. Support-Vector Networks. *Machine Learning*. **20**(3), pp.273–297.

Elish, M.O., 2009. Improved estimation of software project effort using multiple additive regression trees. *Expert Systems with Applications*. **36**(7), pp.10774–10778.

Fedotova, O., Teixeira, L. & Alvelos, A.H., 2013. Software effort estimation with multiple linear regression: Review and practical application. *Journal of Information Science and Engineering*. **29**(5), pp.925–945.

Ferrucci, F. et al., 2010. Using Evolutionary Based Approach to Estimate Software Development Effort. *Evolutionary Computation and Optimization Algorithms in Software Engineering: Applications and Techniques*.

Freund, Y. & Schapire, R.E., 1997. A decision theoretic generalization of on-line learning and an application to boosting. *Computer Systems Science*. **57**, pp.119–139.

Humayun, M. & Gang, C., 2012. Estimating Effort in Global Software Development Projects Using Machine Learning Techniques. **2**(3).

Idri, A., Khoshgoftaar, T.M. & Abran, A., 2002. Can neural networks be easily interpreted in software cost estimation? In *IEEE International Conference on Fuzzy Systems*. pp. 1162–1167.

Jørgensen, M., 2007. Forecasting of Software Development Work Effort: Evidence on Expert Judgment and Formal Models. *International Journal of Forecasting*. **23**(3), pp.449–462.

Khan, M.W. & Qureshi, I., 2014. Neural Network based Software Effort Estimation: A Survey. *Int. J. Advanced Networking and Applications*. **05**(04), pp.1990–1995.

Khatibi, V. & Jawawi, D.N.., 2010. Software Cost Estimation Methods: A Review. *Journal of Emerging Trends in Computing and Information Sciences*. **2**(1), pp.21–29.

Kummer, N. & Najjaran, H., 2014. Adaboost.MRT: Boosting regression for multivariate estimation. *Artificial Intelligence Research*. **3**(4).

MacDonell, S.G. & Shepperd, M.J., 2003. Combining techniques to optimize effort predictions in software project management. *Journal of Systems and Software*. **66**(2), pp.91–98.

Mair, C. et al., 2000. An investigation of machine learning based prediction systems. *The Journal of Systems and Software*. **53**, pp.23–29.

Malhotra, R. & Jain, A., 2011. Software effort prediction using statistical and machine learning methods.

Molokken, K. & Jorgensen, M., 2003. A review of software surveys on software effort estimation. pp.223–230.

DAS Monteiro, J.A., 2002. Multiple Additive Regression Trees a Methodology for Predictive Data Mining for Fraud Detection.

Nayebi, F., Abran, A. & Desharnais, J.-M., 2015. Automated selection of a software effort estimation model based on accuracy and uncertainty. *Artificial Intelligence Research*. **4**(2), p.p45.

Oliveira, A.L.I., 2006. Estimation of software project effort with support vector regression. *Neurocomputing*. **69**(13-15), pp.1749–1753.

Prabhakar, 2013. Prediction of Software Effort Using Artificial Neural Network and Support Vector Machine. *International Journal of Advanced Research in Computer Science and Software Engineering*. **3**(3), pp.2277–128.

Prakash, B.V.A., Ashoka, D. V & Aradhya, V.N.M., 2013. An Evaluation of Neural Networks Approaches used for Software Effort Estimation. *Proc. of Int. Conf. on Multimedia Processing, Communication and Info. Tech., MPCIT*. pp.292 – 296.

Radlinski, L. & Hoffmann, W., 2010. On predicting software development effort using machine learning techniques and local data. *International Journal of Software.* **2**(2).

Reyzin, L., 2003. On Boosting Sparse Parities. pp.2055–2061.

Rodríguez-Soria, P., 2010. A Review of Parametric Effort Estimation Models for the Software Project Planning Process. *Seke*.

Schank, R.C., 1982. Dynamic memory: a theory of learning in computers and people.

Schapire, R.E. et al., 1999. A Brief Introduction to Boosting Generalization error. *Ijcai 99*, pp.1401–1406.

Schapire, R.E., 2009. A Short Introduction to Boosting. *Society*. **14**(5), pp.771–780.

Schapire, R.E., 2013. Explaining adaboost. *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*. pp.37–52.

Schoedel, R., 2006. PROxy Based Estimation (PROBE) for Structured Query Language (SQL). *Management*, (May).

Shivhare, J. & Rath, S.K., 2014. Software effort estimation using machine learning techniques. *Proceedings of the 7th India Software Engineering Conference on - ISEC '14*. pp.1–6.

Singh, Y., Bhatia, P. & Sangwan, O., 2007. A review of studies on machine learning techniques. *International Journal of Computer Science and Technology*. **1**, pp.70–84.

Smola, a J. & Schölkopf, B., 2004. A tutorial on support vector regression. *Statistics and Computing*. **14**, pp.199–222.

Wen, J. et al., 2012. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*. **54**(1), pp.41–59.

Other Publications

Peterson, L., 2009. K-nearest neighbor. *Scholarpedia*, **4**(2), p.1883. Available at: http://www.scholarpedia.org/article/K-nearest_neighbor [Accessed February 28, 2016].

Sayyad Shirabad, J. and Menzies, T.J., 2005. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada.